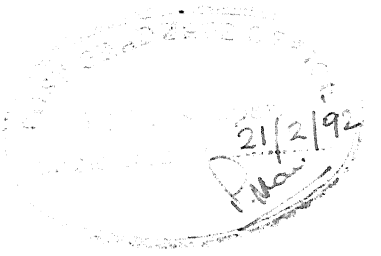
HEURISTIC FOR COORDINATED SCHEDULING IN MATERIAL REQUIREMENT PLANNING

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by
S. R. YOGANANDAN

to the
DEPARTMENT OF INDUSTRIAL & MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
FEBRUARY, 1992



CERTIFICATE

It is certified that the work contained in the thesis entitled "Heuristic for Coordinated Scheduling in Material Requirement Planning " by S.R.Yoganandan has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. R.R.K.Sharma)

Assistant Professor

Industrial and Management Engineering

Indian Institute of Technology

Kanpur - 208016.

Febraury,1992.

TK
658.53
Y75h

16 MAR 1992
CENTRAL LIBRARY
Acc. No. 113082

IME-1992-M-YOG-HEU

ABSTRACT

MRP system has to be used effectively to reduce the costs and due date performance. One of the unaddressed main problem in MRP is the uncoordinated arrival of different subassemblies at an assembly centre.

In this study, this problem has been addressed. A multi-stage, multi-product hypothetical production system is simulated to generate data about uncoordination. Then, a heuristic which seeks to achieve a better coordinated scheduling of all parts is developed.

Preliminary research indicates that such a heuristic improves the performance in a significant manner. Particularly, the aggregate lateness of all end items and the actual inventory carrying costs is reduced substantially .

ACKNOWLEDGEMENT

First and foremost, I would like to thank my Guide **Dr R.R.K Sharma** who provided constant reassurance and support, especially when the going was not smooth. Words are not sufficient to express my gratitude towards him,without whose help this work would not have been possible.

I would also like to take this opportunity to thank all the Faculty members, who have made my stay at I.I.T Kanpur an informative and memorable one. I also offer my thank to all my friends, to mention a few Lt.Col. Mahendra singh, Lawania, Vishwas, Deshraj, Mukul, Jha, Siva, Prabhakar, Sreekant, Anu, Piyush, Ajay and Bhajikaye who helped me a lot during the tough times.

I finally thank the I.M.E family for being what it is.

S.R. YOGANANDAN

TABLE OF CONTENTS

L I S T O F F I G U R E S

LIST OF TABLES

CHAPTER 1. INTRODUCTION

Owing to the pressure created by the economic environment and loss of markets to competition from countries throughout the world there is a renewed interest in the theory and practice of material management. High interest rates and disinflation have intensified the strategic role of material management. Additionally, the importance of material management, more often referred to as inventory management, for providing better customer service in terms of minimizing production lead times, responding to the market demand changes and meeting customer delivery dates, producing products at least cost has been heightened by the embarrassing success of foreign competitors.

Seventy percent of the costs involved in producing a product is on materials. Hence, Inventory management has become the **frontier for maximizing profits** in 1990s [15]. In the area of manufacturing inventory management, the most successful innovations are embodied in what has become known as **material requirement planning**, popularly referred as MRP. MRP has become a new way of life in production and inventory management, displaying older methods in general [16]. Inventory management is incomplete unless an MRP system is included as an integral part of inventory management.

MRP is a Production & Operation Management computer information system that dynamically develops a schedule of planned orders of materials in each time period of the planning horizon [15]. Through a set of logically related records, procedures, and rules for decision making, MRP translates a master production schedule (MPS) into time-phased net requirements for each inventory item needed to implement the schedule and spells out the plan for covering these requirements. The system specifies materials to be procured and the actions to be taken, and determines when they are needed during the planning period.

With MRP, requirements for large raw materials and work-in-process inventories are drastically reduced, and better control exists over quantities and timing of deliveries of raw materials, sub assemblies, and assemblies to production. These controls help to reduce labour, inventories, and overhead costs.

1.1. Scope of the present work

MRP has become a popular buzz-word which has impressed business associates. MRP is used throughout industry. But, to be in the competitive edge over other companies MRP has to be used effectively. It has to be modified to address the ground realities of the environment and to keep the capital blocked in the shop floor as work-in-process inventory as low as possible.

It is the contention of Dr. Eliyahu Goldratt, founder and inventor of optimized production technology (OPT), that the management of a manufacturing organization is made easier if efforts are directed toward the control of all *critically constraining resource* (CCR). On shop floor of multi-stage, multi-product system using MRP, these CCRs are coordination of all levels of sub assemblies, components, and parts.

This motivated to study the MRP system and to address this uncoordinated scheduling and to find the ways, if any, to reduce work-in-process inventory and improve the delivery schedules. The objective of this study is as follows:

1. To develop an MRP/Production system simulation model.
2. To use this model
   (i) To study uncoordinated behavior of components and parts.
   (ii) To find some ways to attain better coordinated scheduling.

1.2 Organisation of the Thesis

Chapter two takes into the world of MRP and gives a helicopter view over all concepts,assumptions,mathematics of MRP and also shortcomings of MRP.

Chapter three provides a glance of literature survey, in the area of MRP, scheduling and multi-product assembly system that was done before starting the work.

Chapter four describes the new heuristics developed and the idea behind it. This chapter also discusses the various performance criteria considered to evaluate the heuristics.

Chapter five provides the description of the system being modeled in the present study, with associated assumptions and the performance criteria used to evaluate the system.

Chapter six outlines the general features about the implementation of simulation model, with some details about the working of different modules.

Chapter seven describes the study that was performed and the statistical analysis carried out on a set of eight sample problems. The detailed result of one sample problem and the summarized results of all eight problems are tabulated.

Chapter eight discusses the conclusion arrived from the study and the scope for future work.

CHAPTER 2. AN OVERVIEW OF MRP SYSTEM

This chapter provides a general overview of Material Requirement Planning (MRP) system. This chapter mainly deals with the objectives, purpose, assumptions, and prerequisites of MRP, its inputs and outputs, and its role in manufacturing, planning & control system. We briefly review its working, mathematical models available for analysis of the decisions to be taken that greatly affect the system performance.

MRP is a formal, computerized information system that integrates the scheduling and control of materials. Through a set of logically related records, procedures, and rules for decision making, MRP translates a master production schedule (MPS) into time-phased net requirements for each inventory item needed to implement the schedule and spells out the plan for covering these requirements. The system specifies materials to be procured and the actions to be taken, and determines when they are needed during the planning period.[15]

2.1. MRP objectives

MRP is necessary because of the volume of materials, supplies, and components involved in producing a company's product line and the speed with which management needs to react to the constant dynamic changes of the economy. An MRP system has the following objectives:[15]

* Improve customer service, thereby increasing sales and lowering prices
* Reduce inventories
* Provide a change-sensitive, reactive manufacturing system

* Improve efficiency through:
    * Reduced idle time.
    * Reduced setup costs.
    * Avoidance of unplanned delays.
    * Fewer cancelled orders and changes in order quantities.
    * Information for schedule planning before actual release of orders.
* Aiding in capacity planning.

2.2. Purpose of the MRP system

The main purpose of an MRP is to control the inventory levels, assign appropriate priorities, and plan capacity of the manufacturing system. Controlling inventory means ordering the right part at the right quantity and time. While assigning priorities among different assemblies it is required to consider due dates and inventory carrying costs. Capacity planning is to decide on machine capacities with a reasonable capacity utilization for a schedule of due dates and throughput. Thus an MRP system is designed to answer the question of what *need* be produced (i.e., what capacity is required and what is to be subcontracted) to meet a given master production schedule.

2.3. Assumptions & Prerequisites of MRP

MRP is effective for companies with the following production characteristics of manufacturing operations:

* Products are produced by using a defined sequence of materials

(components, parts and sub-assemblies).

* A dependent demand exists.
* Demand for components is variable and/or discontinuous in nature.

Proper use of an MRP is based on following assumptions and prerequisites:

* The final product consists of component parts that are uniquely identified.
* A master production schedule exists, and is stated in bill-of-material terms.
* A computerised bill-of-materials exists at planning time.
* Inventory records provide data on the status of each item.
* Lead times are known for the individual items.
* File data are accurate and up-to-date.
* All components go in to and out of inventory stock.
* Discrete dibursement and usages of component material is required.
* Component parts for the final product are needed at the time of assembly order release.
* Independent processing of manufactured items is done.

The above requirements are critical for setting up an MRP. Implementing an MRP requires that MRP data files, shop floor control, and data integrity be well established. In addition, the bill-of-materials files must be computerised, which is itself a major task.

2.4.Role of MRP in Manufacturing Planning and Control Systems

The field of manufacturing planning and control is concerned with management problems and techniques for their solution, as well as with the linkages or interactions among particular problem areas. This full system

includes all data inputs, system modules, and feedback connections. The front end section of the MPC system is the set of activities and systems for the overall direction settings, which produces master production schedule (MPS). The back end or execution section of the system deals with the detailed scheduling of the factory and with managing materials coming from vendor plants.

MRP is the central system in the engine portion. It has the primary purpose of taking a period-by-period (time-phased) set of master production schedule requirements and producing a resultant time-phased set of components/raw material requirements. In addition to MPS inputs, MRP has two other basic inputs. A bill of material shows, for each part number, what other part numbers are required as direct component. The second basic input to MRP is inventory status.

The MRP system serves a central role in material planning and control. It acts as a translator of the overall plans for production into the detailed individual steps necessary to accomplish those plans. It provides information for developing capacity plans, and it links to the systems that actually get the production accomplished. Due to this, the MRP process is where many companies have chosen to start the development of their formal MPC systems.

2.5. The mechanics of MRP

Material Requirement Planning is a technique or a set of systematic procedures for managing inventory in a manufacturing operation. There are four basic steps to the MRP process:

1. net
2. lot

3. offset

4. explode.

Netting is subtracting of on-hand and on-order quantities, from gross requirements. *Lotting* is the determination of individual batch or order quantities for manufacturing based on the calculated time-phased net requirements. There are many lot sizing models or techniques which will be discussed later. Some of the techniques involve economic balancing of set-up and inventory carrying cost and others are simpler rules such as using a fixed number of period requirements. *Offsetting* is the determination of the appropriate time for release of planned orders so that they will be completed in time to satisfy demand. The planned order release data is determined by subtracting the lead time from the date of the earliest net requirement which it is intended to satisfy. *Explosion* is the calculation of gross requirements for the next lower level components which result from planned orders. Explosion may take into consideration shrinkage or yield of the components in the manufacture of the higher level item and the the quantity of each component required to make the higher level item. The "quantity per" factors for the components are multiplied by the planned order quantities for the parent item and then, if appropriate, the result is divided by the yield factor for the component to give the requirements for the components .

The process of determining net requirements, lot sizing, and off-setting of planned orders then performed for the lower level components. If the lower level components are also manufactured items, their planned orders are exploded into gross requirements for their components and so on until planned orders are generated for purchased parts and materials.

In an on-going situation, each MRP planning cycle reevaluates and

adjusts the plans generated in the previous cycle. It is this ability to reevaluate plans in light of changing conditions and signal appropriate corrective actions which makes MRP such a valuable tool in manufacturing control. This adjustment capability is an important part of the MRP methodology. Frequent adjustment procedures cause frequent changes in plants which causes "nervousness" also.

2.6. Lot sizing in MRP

In MRP, lot sizing is activity of translating into an order quantity the net unfiled requirements for a given item (while ignoring scheduling decisions). Order quantity is a question of minimizing the costs of ordering (or setup) plus the cost of carrying the item in stock. Certainly, it is desirable in MRP to generate the least-cost purchasing and manufacturing orders.

There are many approaches to selecting the least-cost order quantity, which are discussed extensively in the MRP literature.(Orlicky, 1975; Berry, 1972; Wagner-Whitin, 1958; Dematteis, 1968; Gorham, 1968; Silver-Meal, 1973). Among these approaches, Wagner-Whitin algorithm is an optimizing technique based on dynamic programming model. It is popularly used as it has been proved to be computationally efficient. It is described below in brief for the sake of completeness.

2.6.1. Wagner-Whitin Algorithm

Wagner-Whitin algorithm [25] evaluates all possible ways of ordering to cover net requirements in each period of the planned horizon. Its objective is to arrive at the optimum ordering strategy for the entire net requirements schedule. The Wagner-Whitin algorithm is "elegant" in that it

reaches this objective without actually having to consider, specifically, each of the strategies that are possible. The algorithm is as follows :

Let d_t= quantity needed,

i_t = interest charge per unit of inventory carried forward to period t +1;

s_t= ordering (or setup cost)

x_t= quantity ordered (or manufactured)

where t denotes period, t = 1,2,...,N.

It is assumed that all period demands and costs are non-negatives. The problem is to find a program $x_t >= 0, t = 1,2,...,N$, such that all demands are met at a minimum total cost;

$$\text{Let } F(t) = \min \left[\min \left[s_j + \sum_{h=j}^{t-1} \sum_{k=h+1}^{t} i_h \, d_k + F(j-1) \right], s_t + F(t-1) \right]$$

where F(1) = s_1 and F(0) = 0. That is, the minimum cost for the first t periods comprises a setup cost in period j, plus charges for filling demand d_k , $k = j + 1,...,t$, by carrying inventory from period j,plus the cost of adopting an optimal policy in periods 1 through j -1 taken by themselves.

The Algorithm at period t^*, $t^* = 1,2,...,N$, may be stated as

1. Consider the policies of ordering at period t^{**}, $t^{**} = 1,2,...,t^*$, and filling demands d_t, $t^* = t^{**},t^{**}+1,...,t^*$, by this order.

2. Determine the total cost of these t^* different policies by adding the ordering and holding costs associated with placing an order at period t, and the cost of acting optimally for periods 1 through t^{**} -1 considered by themselves. The latter cost has been determined previously in the computation for periods $t^* = 1,2,...,t^*$ -1.

3. From these t^* alternatives, select the minimum cost policy for periods 1 through t^* considered independently.

4. Proceed to period $t^* + 1$ (or stop if $t^* = N$)

2.7. Sequencing in MRP

MRP provides component due dates by the expected lead time from each parent item. Once the due dates have been determined, it is left to the shop floor control (SFC) system to meet these deadlines. The sequencing rule is used to select the next job to be processed from a set of jobs awaiting service. Sequencing rules are normally used to minimize total inventory cost. The nature of the dispatching rule employed influences delay costs, in-process and final inventory costs, and setup costs. Due date based sequencing rules are the best performers in the multi stage job shop [5]. Critical ratio rule (CRR), a member of the family of due date based rules, performs well in MRP environment [8].

2.7.1. Critical Ratio Rule (CRR)

A good discussion of several critical ratio rules (including some in actual use at Black and Decker, Hughes Archoff, and Western electric) is provided by Putnam et al [18]. In its most general form, it is computed as

$$\text{Critical Ratio} = \frac{[\text{time remaining} - \text{work remaining}]}{[\text{work remaining}]}$$

If CRR value of a job is less than one, then, it is behind the schedule and it has to be given high priority. If CRR value is greater than one, then, it is ahead of schedule and it can be delayed. Simply, it means, the least CRR value, higher the priority and vice versa.

2.8. Shortcomings of MRP

Many of the assumptions made in MRP are unrealistic. In this section, we will discuss some of these assumptions, the problems that arise as a result of them, and the means for dealing with these problems.

2.8.1.Uncertainty

Underlying MRP is the assumption that all required information is knownwith certainty. However, uncertainties do exist. The two key sources of uncertainty are the forecasts for future sales of the end item and the estimation of the production lead times from one level to another. Safety stock can be included to protect against the uncertainty of demand. The manner in which the uncertainty transmits itself through a complex multi level production system is not well understood. For that reason, it is not recommended to include independent safety stock at all levels of the systems. Rather suitable safety levels can be built into the forecasts for the end item. These will automatically be transmitted down through the system to the lower levels through the explosion calculus.

2.8.2. Capacity Planning

Another important issue that is not treated explicitly by MRP is the capacity of the production facility. Capacity requirement planning (CRP) is the process by which the capacity requirements placed on a work center or group of work centers is computed by using the output of the MRP planned order releases. If the planned order releases result in an infeasible requirements schedule, there are several possible actions. One is to

schedule overtime at the bottleneck locations. Another is to revise the MPS so that the planned order releases at lower levels can be achieved with the current system capacity. This is clearly a cumbersome way to solve the problem, requiring an iterative trial and error process between the CRP and the MRP.

2.8.3. Rolling Horizon and System Nervousness

In practice, production planning environment is dynamic. The MRP system may have to be rerun each period and the production decisions are reevaluated. Often it is the case that only the lot sizing decisions for the current planning period need to be implemented. *Rolling horizon* is used to refer the situation in which only the first-period decision of an N-period problem is implemented. The full N-period problem is rerun each period to determine a few first-period decision.

Another common problem that results when using MRP is *nervousness.* The term was coined by Steele [23], who used it to refer to the changes that can occur in a schedule when the horizon is moved forward one period. Some of the causes of nervousness include unanticipated changes in the MPS because of updated forecasts, late deliveries of raw materials, failure of key equipment or absenteeism of key personnel, and unpredictable yields.

2.9. Factors affecting the computation of requirements

The computation of requirements is complicated by six factors:

1. The structure of the product, containing several manufacturing *levels* of materials, component parts, and subassemblies

2. *LOT sizing,* i.e.,the ordering of inventory items in quantities exceeding net requirements, for reasons of economy or convenience

3. The different *individual lead times* of inventory items that make up the product

4. The timing of the end-item requirements across a *planning horizon* of, typically, a year's span or longer, and the recurrence of these requirements within such a time span

5. Multiple requirements for an inventory item due to its so-called *commonalty,* i.e., usage in the manufacture of a number of other items

6. Multiple requirements for an inventory item due to its *recurrence* on several levels of a given end item.

CHAPTER 3. PREVIOUS RESEARCH

The recent growth in the use of material requirement planning (MRP) systems has resulted in increased interest in the topic of decision making in multi-stage, multi-product systems. MRP has been subjected to extensive research. Particularly, there is hardly a scarcity of research in the area of lot sizing and dispatching rules, their interaction and their impact of MRP system.

Biggs.J.R.[4] reported that the effect of lot sizing and sequencing rules and the interaction between these is significant at the level of 0.01 significance. He has argued that a particular sequencing rule may work in opposite direction to particular lot sizing rule or vice versa.

Blackburn.J.D., and Millan.R.A. [7] have modeled a multi-stage problem analytically to indicate the potential errors inherent in the commonly purposed single-pass, stage-by-stage approaches. Then, based on this analysis, several simple cost modifications are suggested.

Biggs.J.R.[5] has studied the effect of using various priority scheduling rules to capacity management and shop floor control in a multi-stage, multi-product system which uses MRP. He found that Critical Ratio Rule (CRR) performs consistently well over different levels of capacity.

Grasso and Taylor [11] studied main effects and interaction effects among four factors, namely, lead time, buffering alternatives, lot size rule and cost value on total cost and reported that main effects of each factor and the interaction between lead time distribution and lot size and that between buffering alternative and cost value on total cost are significant.

South and Steward [22] report their experience at Hix corporation wherein fifty percent increase in lot sizes and planned manufactured lead time resulted in low system tardiness without increase in WIP inventory

costs.

As already discussed, Lot sizing, sequencing, determination of planned lead times and capacity planning are important decisions in the effective use of MRP. There is a lot of literature available on scheduling.

Panwalkar.S.S., Iswander.W. [17] have classified over 100 scheduling rules and made an attempt to explain the general idea behind different rules.

Blackstone ,Philips and Hoggs [8] have discussed the state of the art in the study of dispatching rules, listed 34 dispatching rules and compared them. They have concluded that no single dispatching rule developed so far consistently produce lower total cost than all other rules under a variety of shop configurations and operating conditions.

Goodwin,J.S., and Goodwin,J.C., [13] have shown that the performance of various sequencing rules do not generalize the multi-stage job shop.

In a recent study, Russell and Taylor [20] have evaluated the performance of various dispatching rules in two different product structures. The authors constructed a BOM representing a tall product structure having four levels and a BOM representing a flat product structure having three levels. They concluded that as the BOM gets taller, the probability that the job will finish after its due date increases.

In another study Goodwin and Weeks [13] have studied the effect of sequencing rule and frequency of priority updating. They concluded that due date based sequencing rules were the best performers in the multi-stage job shop.

Fry, Philipoom and Markland [12] have evaluatedthe performance of selected dispatching rules in an unbalanced multi-stage job shop to determine whether the performance is consistent for jobs which are rated through the bottleneck and for jobs which by-pass the bottleneck. They concluded that sequencing rules which are based on job due date tends to be

the overall better performers. They also concluded that the results for jobs which pass through the bottleneck were not the same for jobs which by-pass the bottleneck.

Potty.V.S. [18] has reported that the choice of a lot size rule has significant impact on the total costs and the choice of sequencing rules has significant effect on number of stock outs and unit stock outs. He also reported that lot sizing using Wagner-Whitin algorithm and sequencing using CRR rule does well in MRP environment.

In a recent study, Sharma.R.R.K., and Potty.V.S. [21] have presented a multi-pass heuristic which improves the performance on the number of stock outs by an iterative interaction with the lot size rule under fixed capacity.

But, Sequencing in a multi-stage, multi-product system requires a coordinated flow of materials throughout the various stages to assure components needed for assembly are available when planned. Coordination is vital since shortage of a single item can halt the assembly process, resulting in sharply diminishing productivity and increased inventory costs. But, surprisingly, no work has been done in this area to question the *uncoordinated scheduling.*

CHAPTER 4. HEURISTICS DEVELOPED

Normally, single pass heuristics are used in MRP systems for simplicity. That is, a single lot sizing procedure is applied to determine the lots and next a single scheduling heuristic is applied. This procedure is popular due to its simplicity. But such a procedure lacks communication between the lot sizing module and scheduling module in various respects. Biggs[4] has established that in fact lot sizing and sequencing rules interact. Hence, there may be scope to improve the solutions by using multi pass heuristics. One such effort can be found in the work by Sharma and Potty[21], where the inventory holding costs were increased for items or assemblies which spends substantially more time in the shop than the theoretical holding costs.

Here, we note that possibility exists to improve lateness/tardiness performance if variance of the arrival times of different subassemblies at an assembly centre can be reduced. This means, in other words, that efforts made to improve the co-ordinated arrival of jobs at various assembly centres could be fruitful. With this preliminary hypothesis we developed a heuristic which seeks to achieve a better co-ordinated arrival at assembly centres. This heuristic is described below.

4.1. Heuristic 1

This heuristic is based on the following idea.

i). Find the components which arrived very late and forced assemblies to wait. Expedite it by increasing its criticallity.

ii). Find the components which arrived very early and waited for a

long time. Delay it by decreasing the criticallity.

The steps of heuristics are described below.

Let T = length of the planning horizon.

BestSol = Best solution arrived at w.r.t aggregate lateness.

R = set of components which has to be expedited.

D = set of components which has to be delayed.

1. Initialization. Set E = { } ; D = { } ;

BestSol = α ;

2. Read input data.

3. Simulate MRP system for the time horizon considered.

4. Find the components which have highest aggregate waiting time and let *j* be that component. E = E + {j} ;

5. Find the components which have less aggregate waiting time and let k be that component. D = D + {k} ;

6. If the current solution is better than the BestSol w.r.t aggregate lateness, then update BestSol = current solution.

7. Increase the criticallity of all components in set E in steps of 0.1.

8. Decrease the criticallity of all components in set D in steps of 0.1.

9. If the number of iterations considered is less than twenty then go to step 3 else print final results.

An increase in the criticallity of a component by increasing the CRR-value, it gets high priority and arrives at early period. The opposite is the case for decreasing CRR-value. So, the aggregate lateness is reduced and the coordination is expected. Actual inventory carrying cost is also expected to reduce. For the sake of comparing the performance of the above heuristics with other multi pass heuristics, we reproduce a heuristic due to

Sharma and Potty with slight modification.

4.2. Heuristic 2

Let R be the set of parts with difference in actual and theoretical holding costs exceeding 10 %

Let S be the set of parts already considered twice in the iteration procedure.

Let K(j) denotes the number of times the part "j" has been considered in the iterative procedure.

Let BestSol denote the best solution arrived w.r.t the aggregate lateness.

The steps of the heuristic are described below.

1. Initialize the following

   S = { } ; K(j) = 0 for all j .

   BestSol = α ;

2. Read input data

3. Simulate the MRP system for the time horizon considered to prepare the set R.

4. Select part "j" from set R which has the highest percentage difference in theoretical and actual holding costs. Increase the holding cost of parts "j" in steps of 0.2 until the lot size changes.

5. Set K(j) = K(j) + 1;

6. If (K(j) >= 2), then update S = S + { j } ;

7. If the current solution is better than the BestSol w.r.t aggregate lateness than BestSol $\leftarrow$ Current solution.

8. R $\leftarrow$ R - S .

9. If R = { } then stop else go to step 3.

An increase in the holding costs is likely to reduce the lot size and reduce the time taken to process that lot. That means, it arrives to the assembly shop early and get assembled. It is expected that it may reduce the aggregate lateness of all assembled items and the actual inventory costs. But, in the same time , an excessive decrease in lot sizes may increase the number of setups, and adversely affect the no of stock outs and the total costs.

4.3. Heuristic 3

In heuristic 2, there is no effort made to secure co-ordination. It is also possible to reduce the aggregate lateness by

i). Changing the lot size and arrives the best solution using the Heuristic 2.

ii). Then, Finding the components which created the uncoordination and applying Heuristic 1.

It is expected that simultaneous changes in the lot sizes and the priorities may reduce the aggregate lateness and the components are expected to arrive with coordination with one another.

The steps of heuristicis described below.

1. Initialize in the same way as in the case of Heuristic 1 and Heuristic 2.

2. Simulate MRP system

3. Apply the Heuristic 2 and get BestSol.

4. Starting with this IIBestSol , Apply the heuristic 1 and get updated Solution.

4.4. Performance criteria

The performance of the system is analysed using the following representative criteria :

1. Aggregate lateness of all end items.
2. Total number of stock outs, final products.
3. Inventory costs, total system.
4. Total number of setups, total system.

Aggregate lateness of end item is considered as a performance criteria because this lateness is due to the *uncoordinated arrival* of all the components.

When the manufacturing organization are facing a tough competition, their performance is judged on the basis of the punctuality of delivering the parts as ordered. Number of stock outs is considered as a performance criteria.

Inventory carrying costs indicate the extent to which capital has been locked up and has a direct influence on the profits earned.

The number of setups is also important, especially when the change over times from one setup to another setup are significant.

The Simulation model used to evaluate the above heuristics is followed in the next chapter.

CHAPTER 5. SIMULATION MODEL

A computer simulation model of a hypothetical multi-stage, multi-product production system is used to generate data about the uncoordinated scheduling. This chapter provides the description of the system being modelled in the present study, with associated assumptions. Simulation is defined as " the process of creating the essence of reality without ever actually attaining the reality itself". Simulation is selected because

1. The system behaviour under disturbance can be predicted easily.
2. Detailed analysis of a complex material requirement planning system is relatively easy.
3. Simulation provides flexibility in experimental design.

5.1. Logic of the model

The simulation model implements standard MRP logic(Fig 5.1). Although a simulation model, it operates essentially deterministically. Since product structures and routing are fixed, it processes entities in a known manner. Demand by time period for end items are assumed known (but not constant) over a finite horizon.

5.2. Assumptions

The system has been designed under the following assumptions:

1. Demand is deterministic.
2. Man power is of uniform ability.
3. Breakdown times and repair times of the machines are deterministic.

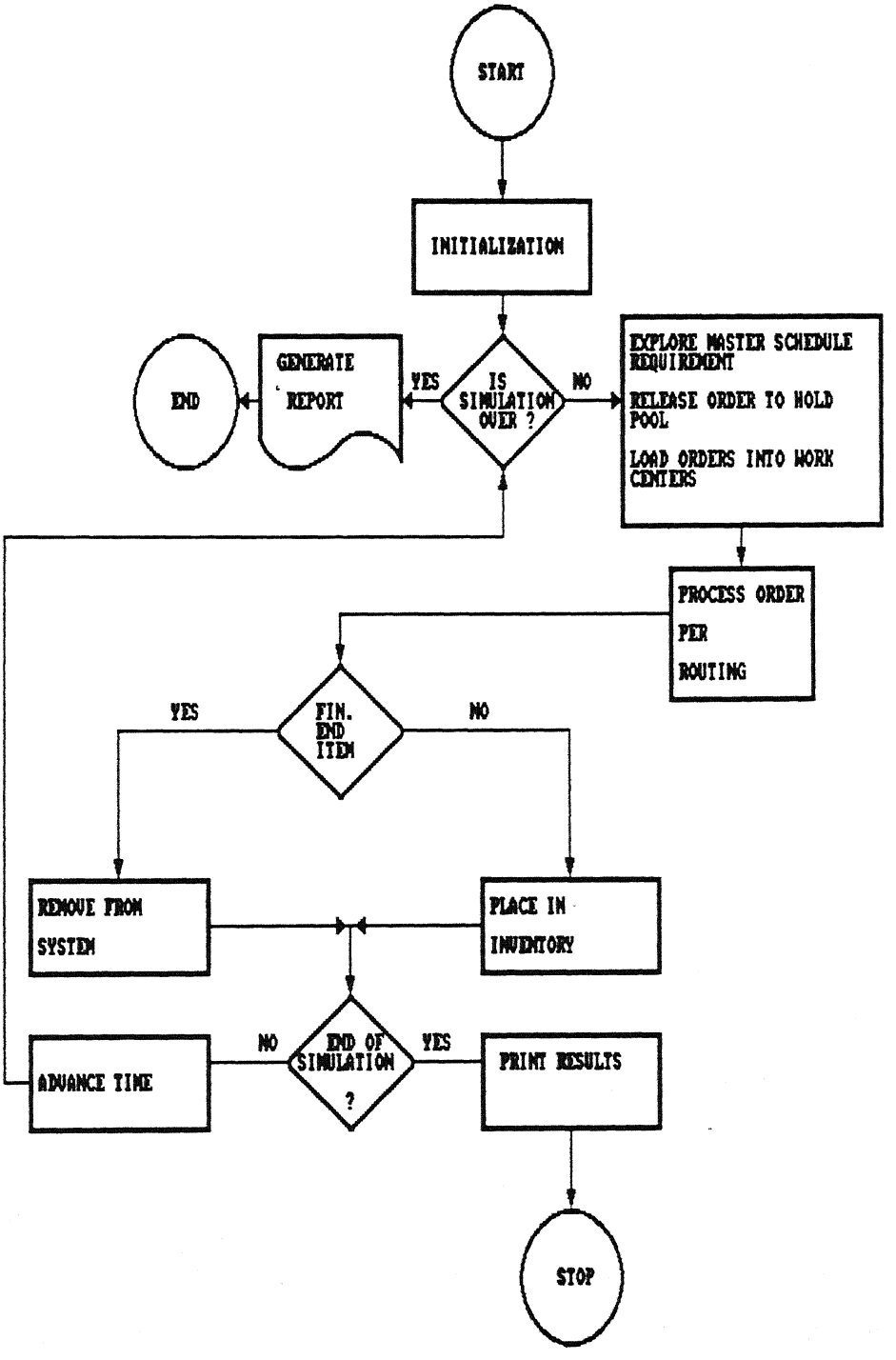


FIGURE 5.1 LOGIC OF SIMULATION MODEL

4. Pre-empting is not allowed.

5. Machine setup and operation times are known.

6. No work center can perform more than one operation at any time.

7. Each lot size, once started must be completed if the component parts are available.

5.3. System description

The entire system is divided into the following five basic modules:

1. Input data module.
2. Planning module.
3. Shop floor control module.
4. Simulation module.
5. Output module.

Input data module:

In this phase, all the input data are read, like, the product structure from Bill-of-material files, external forecasted demand for the end items from the master production schedule file and the inventory status of all items from inventory status file. An inventory status file contains data for each item in inventory, like, scheduled receipts, ordering cost, inventory carrying cost over planning horizon, lead time and quantity on hand

Planning module :

In this module, a detailed exploration of the end items to all subassemblies, components and parts takes place. Then, netting, lot sizing and offsetting takes place. The output of this module is the release of planned orders to the shop floor to start the production operation and to the purchase department to plan purchase orders.

Shop floor control module:

In this module, all the planned orders for a particular period are undertaken for the production. Each part, then, follows their routing pattern over various processing centers. Once the operation gets over in a particular machine, it goes to next machine and waits in the queue. Once the machine is free, the next critical job in the queue will be taken for the processing. Sequencing rule is used to find the criticality of a job. After all processing is over, the transformed part follows to assembly shop , if it has any parent. . Once all necessary components have arrived, the assembly operation starts.

Simulation module:

This module performs the simulation work. It runs and controls the shop floorover the planning horizon considered. The planning horizon consist of forty time buckets.

Output module:

Once the assembly of end item was completed, the end item will be dispatched to the customer. This module provides all the statistics like lateness, number of stock outs, setup costs, inventory carrying cost and total cost incurred over the planning horizon.

5.4. Size of the shop

When Baker and Dzielinski [2] tested some of the dispatching rules in shops of various sizes, they found that size of the shop does not affect the relative performance of the rule. This finding is convenient in that it is cheaper to simulate a small shop. Buffa [9] concludes that: " Since shop size has never appeared as a major variable, it seems that we may be able to experiment with small shops and generalize the resulting conclusion". The

production system used in this study consists of fifteen processing centers and five assembly centers. But, it can be varied.

5.5. Rules considered

Three heuristics discussed in earlier chapter are implemented in the simulation model. Lot sizing considered is Wagner-Whitin algorithm and sequencing rule considered is critical ratio rule.

5.6. Modeling approach

Discrete event simulation with next event scheduling approach is used in this model. In this approach there is no change in the system status between any two consecutive events. The clock time of simulation and the system status are updated at the occurrence of each of the events occurring in the system. In the model that has been developed, the following are the events that causes a change in the system status:

1. Order release at the starting of the period.
2. Processing machine loaded/unloaded.
3. Assembly machine loaded/unloaded.

CHAPTER 6. IMPLEMENTATION OF THE MODEL

The heuristics developed in the previous chapter have been implemented in an MRP environment simulated on a personal computer. This chapter outlines the general features about the implementation, with some details about the working of different modules.

6.1. A word about the implementation

The essential components of the modelsuch as a clock mechanism, data structures to represent transactions, resources, and queues and collection of essential statistics were identified. job. To cope up with this lengthy process of building a simulation model, a number of special purpose simulation languages like Simula, GPSS, Simscript, etc, have been developed and widely used. Although all these special purpose languages have passed through several versions, none of them can be conveniently used to implement the user defined heuristics. Pascal is a highly structured language and often highly sophisticated data structure, and also support dynamic allocation of memory. Because of these reasons data structures of linked lists type can be created to handle the various events, and thus the memory requirements does not depend on the length of the simulation. Because of these reasons, Pascal is used for the implementation of the present situation model.

6.2. Working of different module

In this section, the working of different procedures in our simulation model is described.

6.2.1. Input data module

In this modules three procedures namely, **read_assly_data,**

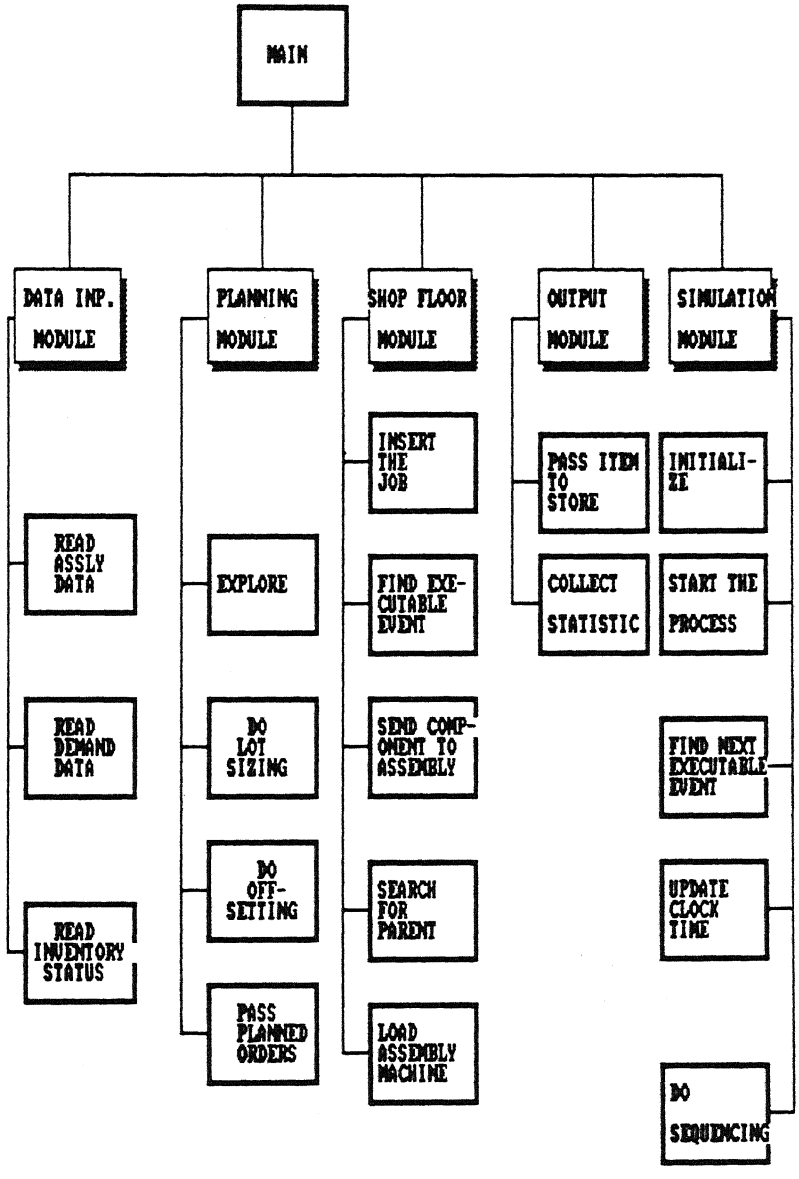


FIGURE 6.1 BUILDING BLOCKS OF MRP SIMULATOR

read_demand_data and **read_inventory_data** are called. **read_assly_data** reads all the details from bill-of-material file. **read_demand_data** reads all the external demands from demand file for all the assembled items. **read_inventory_data** opens the inventory-status file and reads the inventory status, like scheduled receipts, ordering costs and inventory carrying cost over planning horizon , lead time and quantity on hand.

6.2.2. Planning module

In this module, there are three procedures, namely, **do_planning**, **do_lot_sizing** and **explode**. **do_planning** calls the **explode**. **explode** explores all the product structures starting from the end items to all purchased parts to find gross requirements of all sub-assemblies, parts and components. **do_planning** calls **do_lot_sizing** after its complete *netting*. **do_lot_sizing** is the implementation of Wagner-Whitin lot sizing algorithm. Then **do-planning** does *the offsetting* using the lead times of all components. The final output of this module is the detailed planned orders for all components over the time horizon.

6.2.3. Shop floor control module

This module is the important module in the model. It controls all the activities of shop floor from receiving the orders from planning module to sending the final finished products of all end items to finished goods inventory store.

The working of different procedures are described below:

initialise_the_item accepts the planned orders and does the following:

i). If the item needs processing, then, depending on the first machine where the first operation has to be done, the item is inserted in the queue before that machine using **insert**.

ii). If the item doesn't need processing, then, the item goes to assembly machine on which its parent item is going to get assembled. This

job has been done by procedure **search_for_parent.**

insert is a procedure used to insert a item in the queue before the processing machine. In case, the machine is free, the item is loaded immediately using **load_the_processing_machine.**

insert_in_assly_mc receives all the orders for the assembled items from the planning module and the assembling machines in the assembly shop has been informed and asked to keep track of incoming sub-assemblies, components and parts.

search_for_parent is another important procedure in this model. The input to the model is either a finished component or parts in the plant itself or a purchased component from the outside which does not need any processing. The incoming item finds its parent item in the following way.

i). If no_of_parent is one, then, the item has to be placed before the assembly machine where its parent is getting assembled.

ii). If the item has more than one parent item,then the parent item which gets assembled at an early stage gets priority. In case if there are more than one parent item getting assembled at the same period, then the one which needs less quantity gets priority.

load_the_processing_machine is a procedure which will be called whenever the processing operation on a particular machine getscompleted. Then, the finished component is unloaded and updated using **update_the_item.** The next highly critical item is loaded on this machine. The finishing time of this machine is updated.

update_the_item does the following:

Once a particular job is gets completed, then the job has to be sent to the next machine where the next operation has to be carried out. **insert** is called for the above purpose. In case if the item has completed all operations, then, the finished item willbe sent to assembly shop using

search_the_parent.

load_the_assly_mc loads the assembly machine when all the required components are available in required quantity. It also updates the information like finished time, no of stock outs in case if it is late etc.

6.2.4. Simulation module

This is the module which does all the simulation work.

For a particular period, **sequencing_initialisation** does all the necessary initialisations that has to be done at every time period.**start_the_process** starts loading all the processing machine if they free. **find_next_executable_event** finds the next executable event that may occur in the shop and change the system. **update_clock_time** updates the current clock time to time at which the next executable event occurs. **do_sequencing** runs the model for the time horizon considered . In this study, the time bucket consists of fourty time buckets.

6.2.5. Output_module

This module collects all the related statistics using **collect_and_print_statistics.** This also prints the complete result

CHAPTER 7. TEST RUNS & RESULTS

This chapter gives the computational results generated and the statistical analysis carried out on a set of eight sample problems.

7.1. Simulation Run

The simulation was performed over a planning horizon. The planning horizon considered was forty periods. The time period considered was a week. The statistics were collected only after a preliminary run for five periods. The sample problems were selected in such a way that it has all different product structures like flat, tall and mixed. Figures 7.1.1 to 7.1.23 are the different product structures considered in our study. Every problem contains three to five different product structures. Table 7.1.1 gives the details about the product structures contained in eight different problems. Inventory carrying costs from one period to another period and setup (ordering) cost over all periods were considered as variable.

For illustrative purpose, the details of problem no 4 is given fully. The number of end products in this problem is four and total number of parts in this problem is forty five. The subassembly details are given in the table 7.1.2. The work center where the assembly process takes place and the time taken to assemble per unit are given in the table.

The routing details of the parts are given in the table 7.1.3. The numbers within the brackets indicate work centre where the process takes place and the time taken to process per unit. The machining sequence of a part is the same as given in the table.

7.2. Discussion of Results

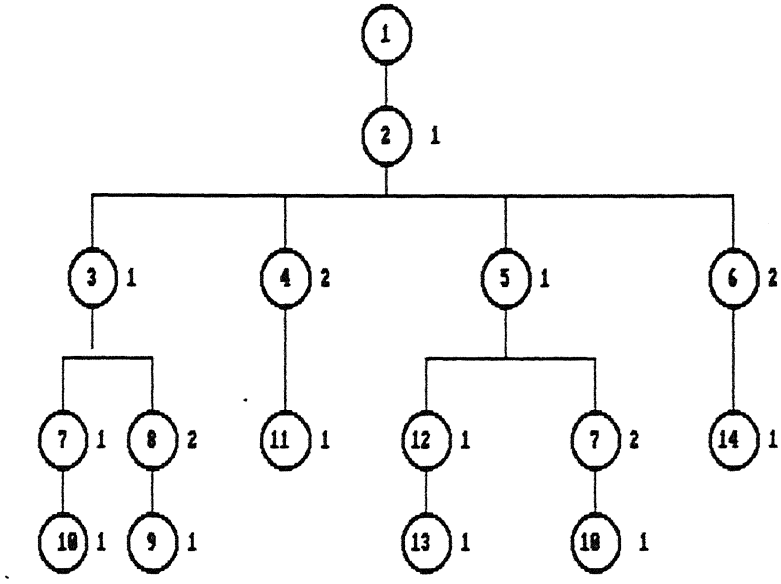


FIGURE 7.1.1. PRODUCT STRUCTURE NO 1

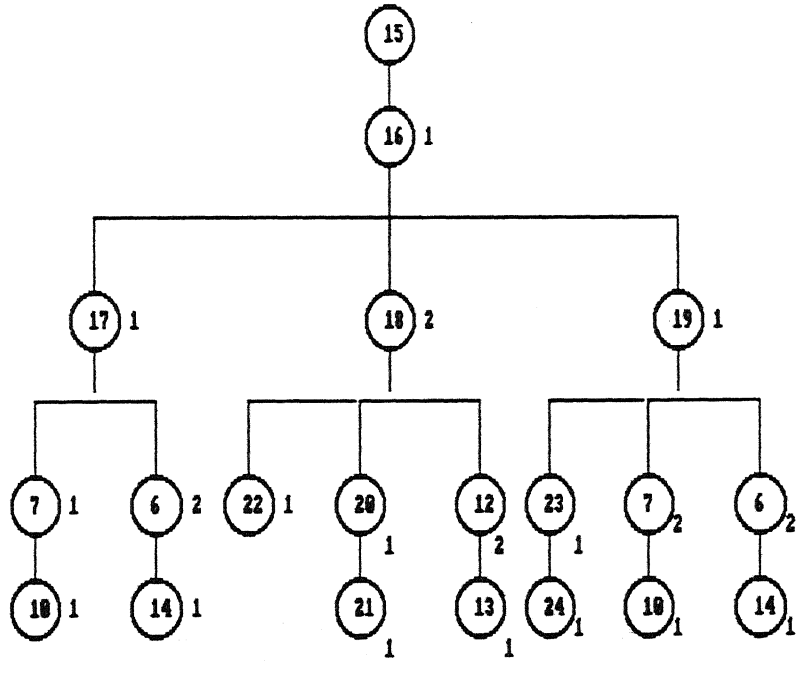


FIGURE NO 7.1.2 PRODUCT STRUCTURE NO 2

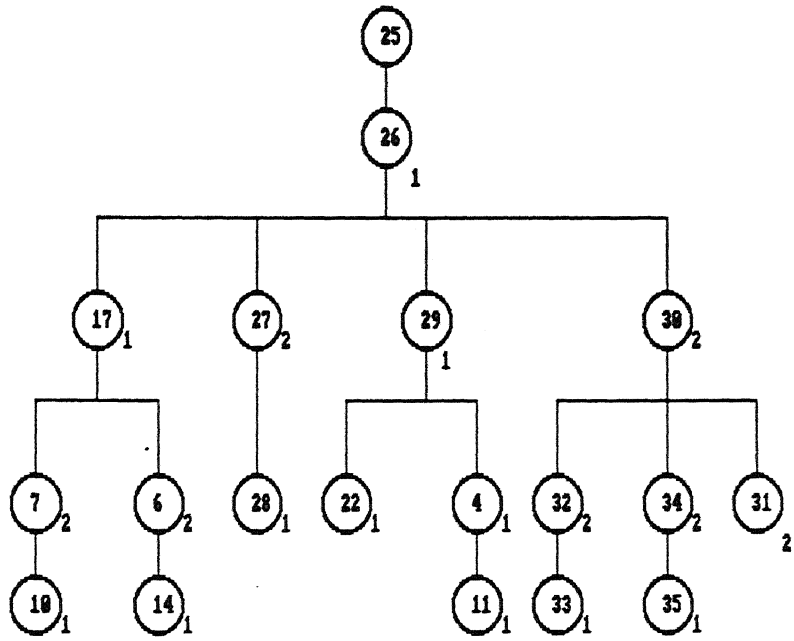


FIGURE NO. 7.1.3. PRODUCT STRUCTURE NO 3

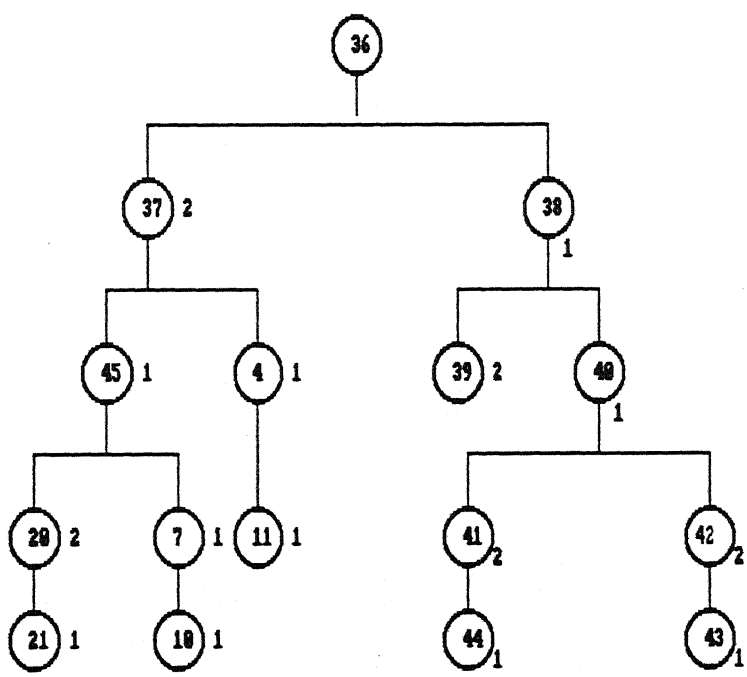


FIGURE 7.1.4 PRODUCT STRUCTURE NO 4

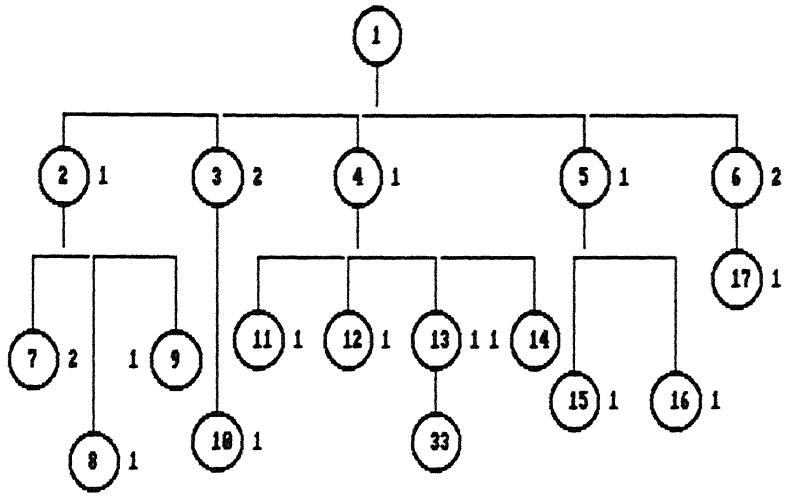


FIGURE 7.1.5 PRODUCT STRUCTURE NO 5

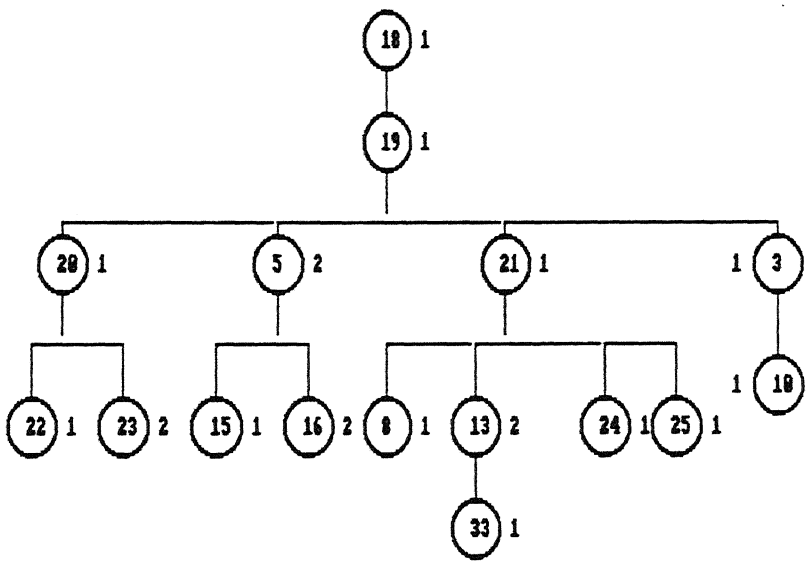


FIGURE 7.1.6 PRODUCT STRUCTURE NO. 6

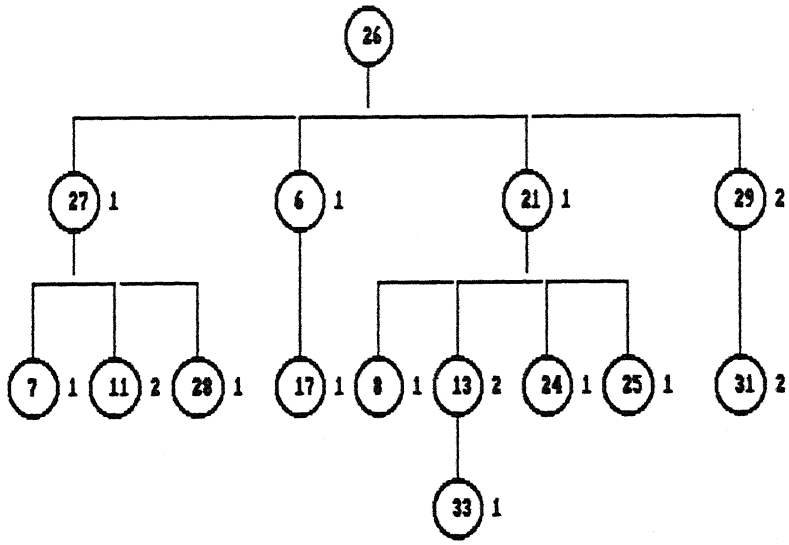


FIGURE 7.1.7 PRODUCT STRUCTURE NO. 7

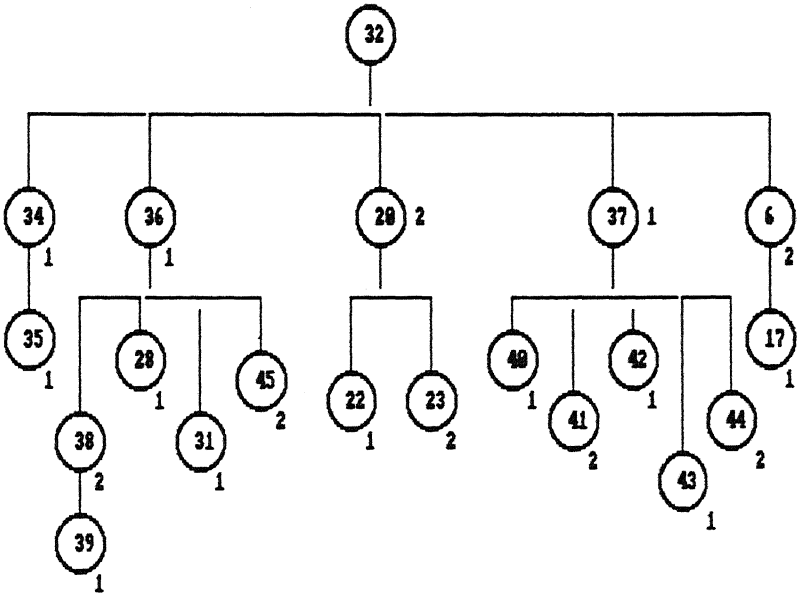


FIGURE 7.1.8 PRODUCT STRUCTURE 8

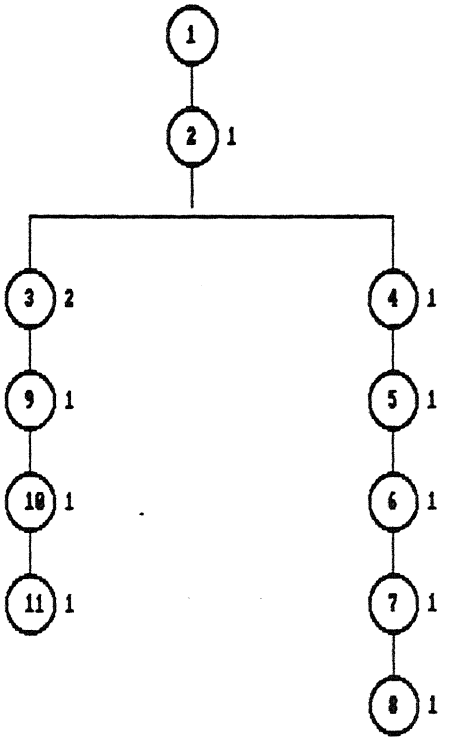


FIGURE NO 7.1.9 PRODUCT STRUCTURE NO 9

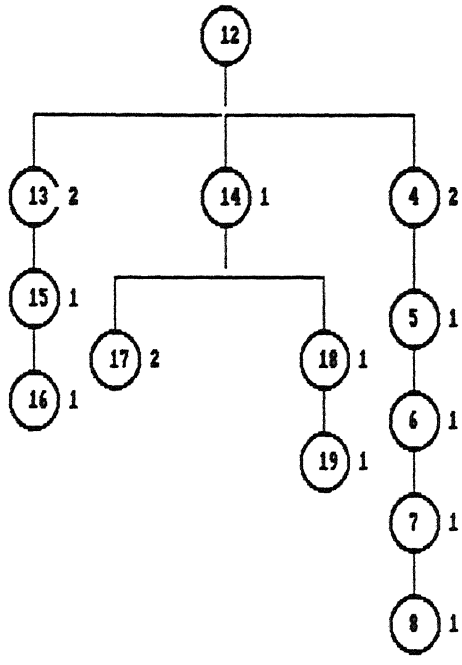


FIGURE NO. 7.1.10 PRODUCT STRUCTURE 10

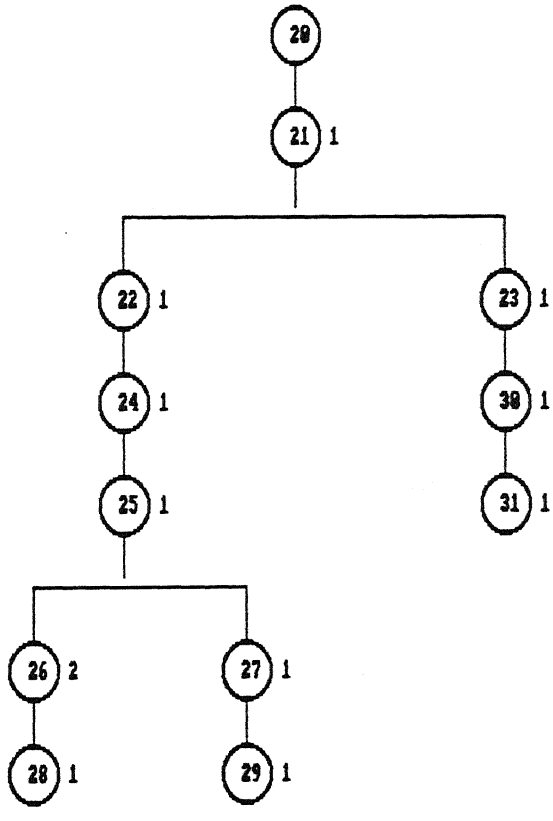


FIGURE NO. 7.1.11 PRODUCT STRUCTURE 11

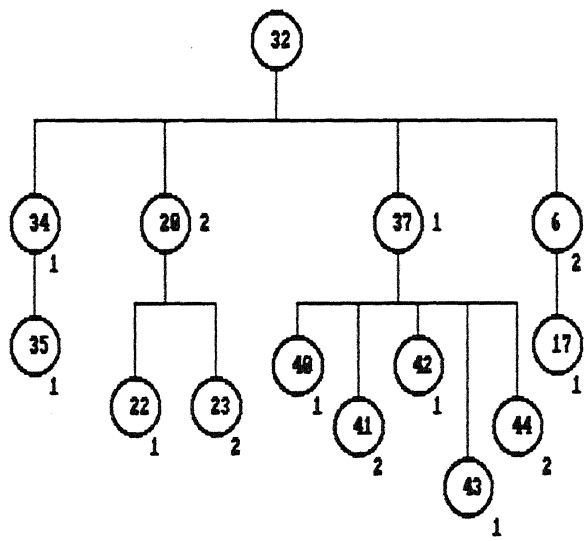


FIGURE 7.1.12 PRODUCT STRUCTURE 12

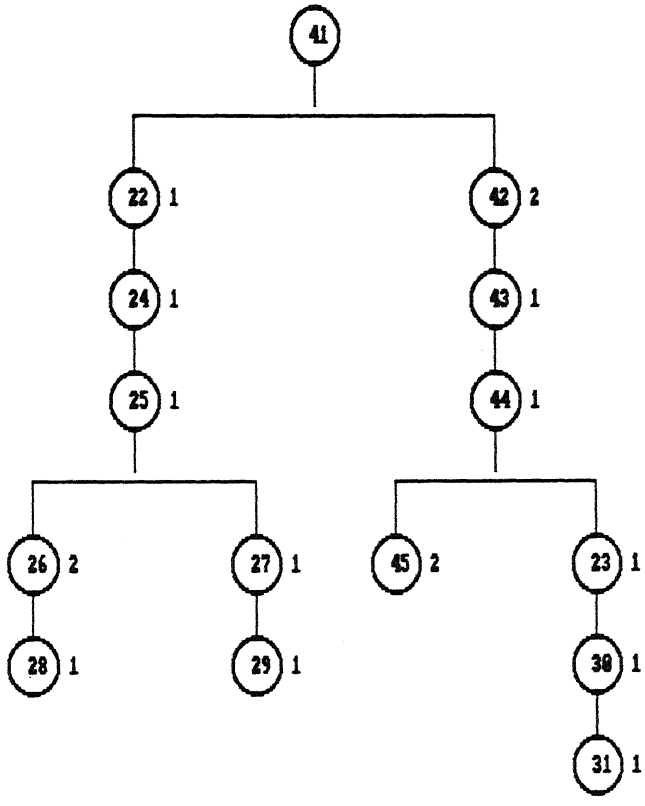


FIGURE NO. 7.1.13 PRODUCT STRUCTURE NO. 13

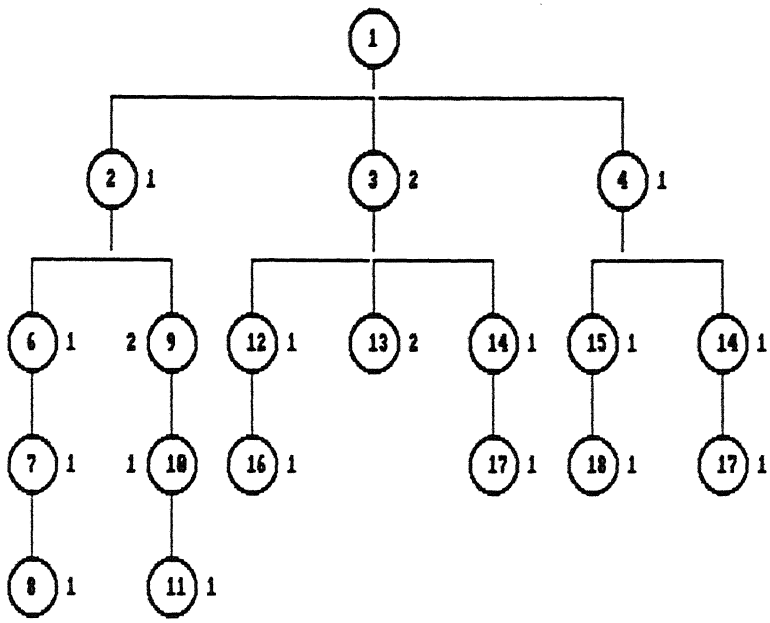


FIGURE NO 7.1.14 PRODUCT STRUCTURE NO 14

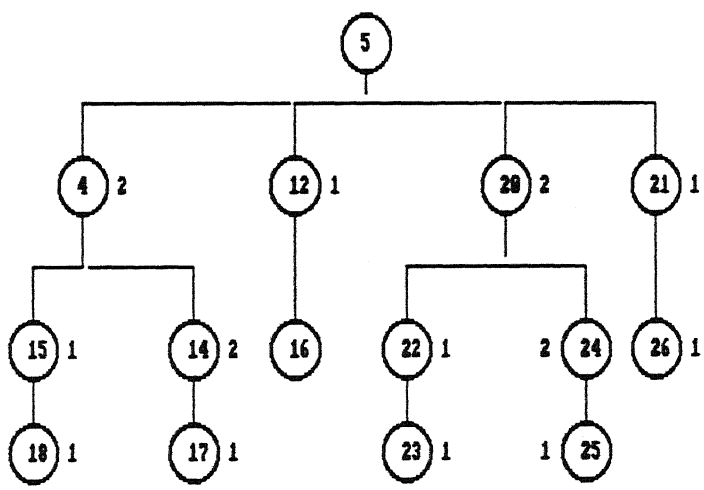


FIGURE NO 7.1.15 PRODUCT STRUCTURE NO 15

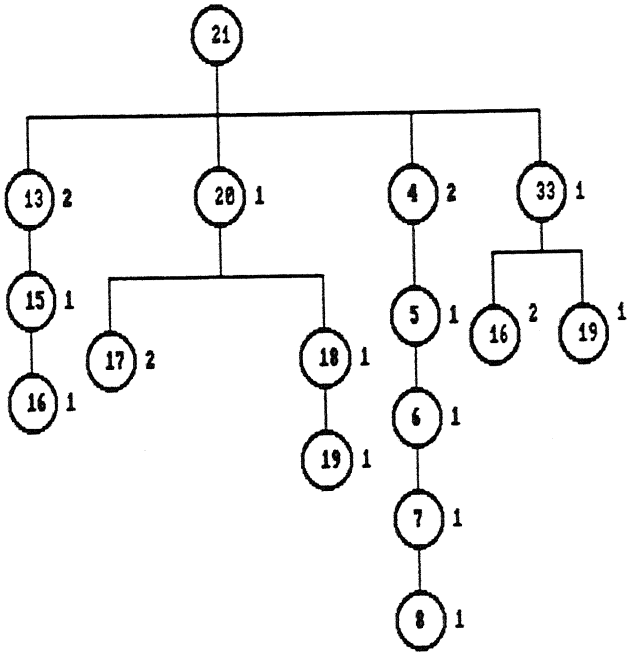


FIGURE NO. 7.1.16 PRODUCT STRUCTURE 16

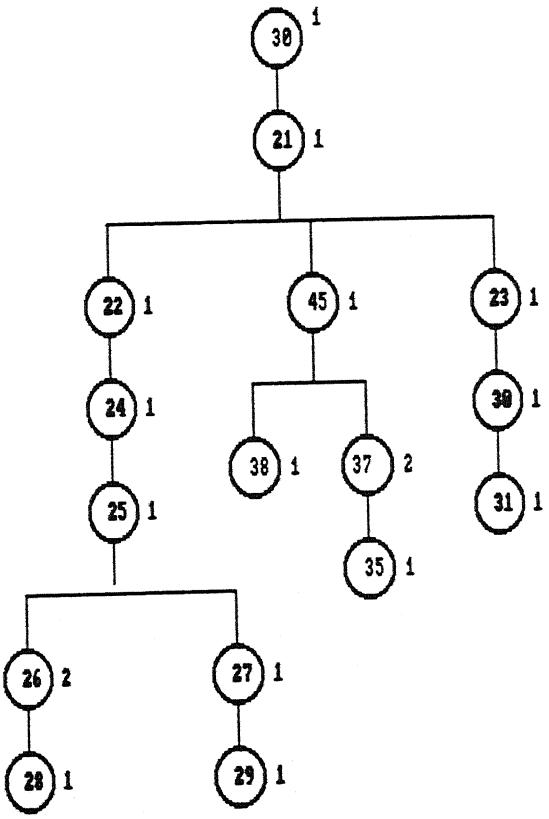


FIGURE NO. 7.1.17 PRODUCT STRUCTURE 17

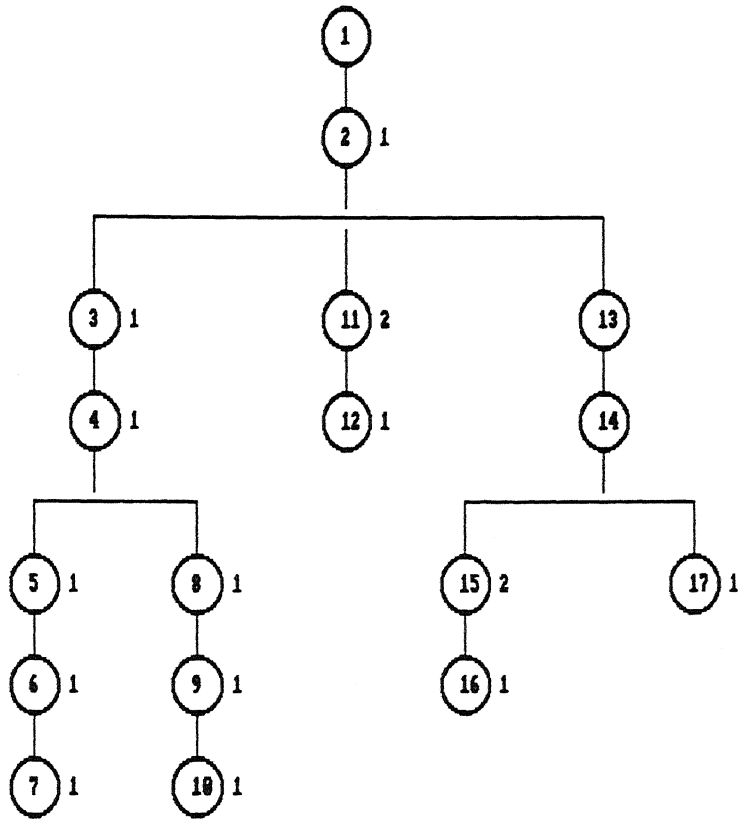


FIGURE No. 7.1.18 PRODUCT STRUCTURE NO 18

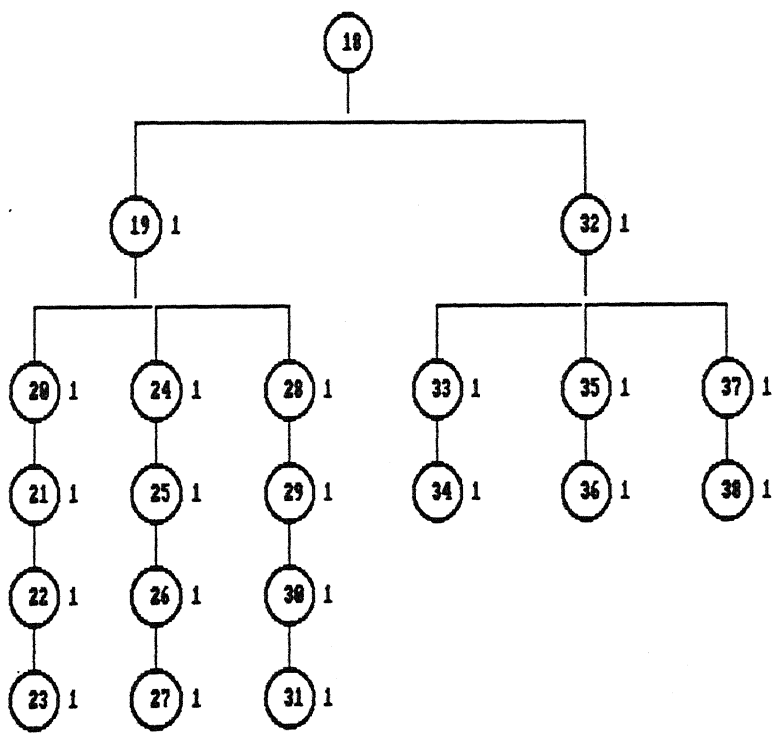


FIGURE 7.1.19 PRODUCT STRUCTURE NO 19

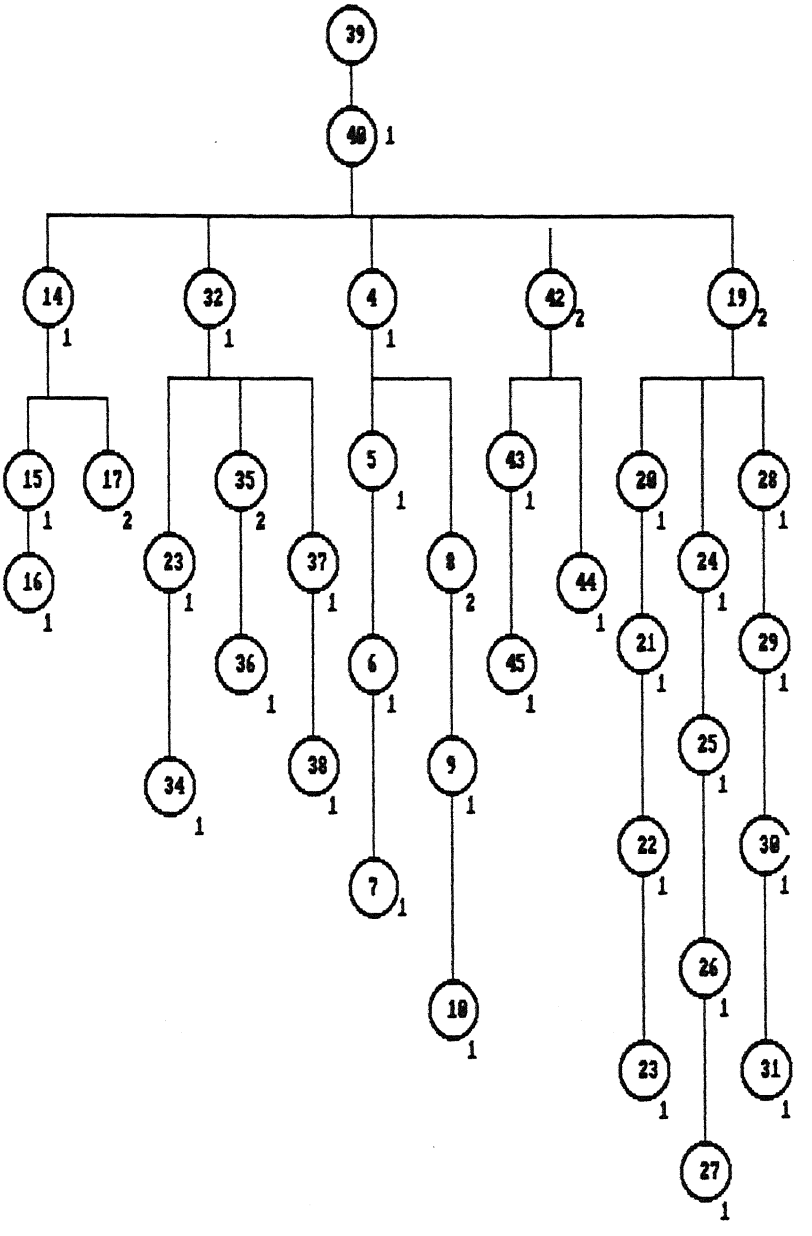


FIGURE NO 7.1.20 PRODUCT STRUCTURE 20

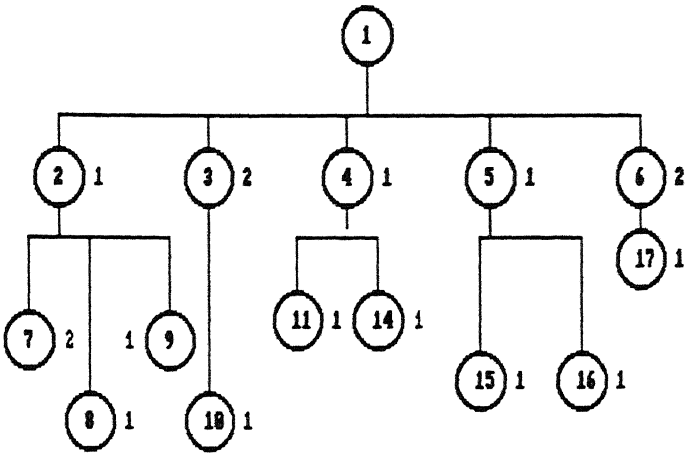


FIGURE NO. 7.1.21 PRODUCT STRUCTURE NO. 21

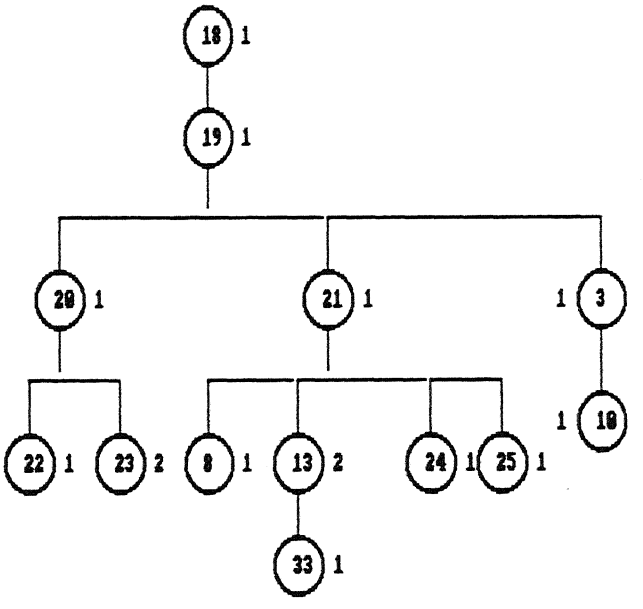


FIGURE NO. 7.1.22 PRODUCT STRUCTURE NO. 22

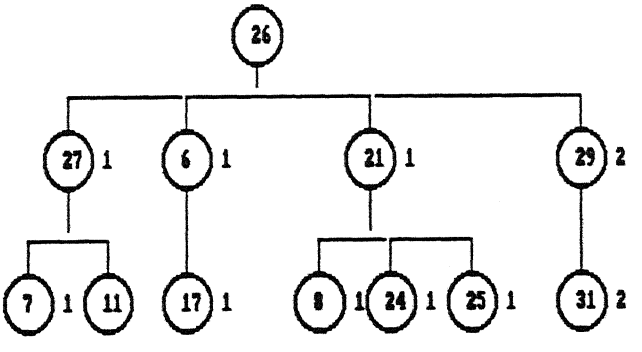


FIGURE NO 7.1.23 PRODUCT STRUCTURE NO 23

TABLE 7.1.1

PRODUCT STRUCTURE OF TEST PROBLEM

PROBLEM NO.	PRODUCT STRCTURE
1	(1, 2, 3, 4)
2	(5, 6, 7, 8)
3	(9, 10, 11, 12, 13)
4	(14, 15, 16, 17)
5	(18, 19, 20)
6	(9, 10, 11, 12, 13)
7	(21, 22, 23,)
8	(9, 17, 11, 18)

TABLE 7.1.2

SUB-ASSEMBLY DETAILS

PART NUMBER	WORK CENTER	TIME PER ASSEMBLY
1	16	40
2	18	45
3	19	30
4	17	25
5	20	15
19	17	20
20	16	30
27	18	34
34	26	20
39	19	45
42	16	15

TABLE 7.1.3
ROUTING DETAILS OF PARTS

PART NUMBER	ROUTING SEQUENCE		
7	(1 , 10)	(13 , 15)	(6 , 9)
8	(14 , 8)	(10 , 7)	(4 , 3)
10	(3 , 20)	(12 , 12)	(9 , 11)
11	(15 , 10)	(1 , 9)	(7 , 7)
16	(5 , 7)	(11 , 21)	(8 , 5)
13	(1 , 5)	(2 , 3)	(9 , 8)
17	(6 , 2)	(10 , 5)	(4 , 13)
18	(2 , 15)	(4 , 17)	(11 , 4)
17	(7 , 25)	(9 , 7)	(2 , 17)
16	(13 , 10)	(6 , 4)	(10 , 6)
23	(8 , 5)	(6 , 11)	(1 , 9)
25	(11 , 10)	(8 , 20)	(12 , 21)
26	(9 , 3)	(5 , 10)	(15 , 15)
41	(8 , 4)	(10 , 19)	(7 , 16)
44	(10 , 10)	(4 , 11)	(14 , 12)
45	(7 , 5)	(11 , 6)	(13 , 8)
33	(11 , 4)	(3 , 3)	(8 , 9)
34	(14 , 8)	(12 , 9)	(3 , 2)
36	(12 , 7)	(2 , 23)	(9 , 15)
38	(15 , 15)	(14 , 19)	(6 , 7)
31	(13 , 25)	(1 , 17)	(5 , 9)
32	(6 , 10)	(15 , 2)	(7 , 10)

Table 7.2.1

SUMMARY OF RESULTS

PROBLEM NO 1

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	8732.61	12.07	42.42	698	105
HEURISTIC ONE SOLN	8005.35	7.62	37.38	698	101
PERCENTAGE IMPROVEMENT	8.32	39.54	11.88	0	4.0
HEURISTIC TWO SOLN	8228.02	9.61	39.68	699	102
PERCENTAGE IMPROVEMENT	5.77	23.39	6.48	-0.14	3.0
HEURISTIC THREE SOLN	8005.0	5.25	35.31	619	101
PERCENTAGE IMPROVEMENT	8.32	58.3	16.76	0.14	4.0

TABLE 7.2.2

SUMMARY OF RESULTS

PROBLEM NO 2

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	605.86	11.82	28.15	791	39
HEURISTIC ONE SOLN	605.86	5.81	22.13	791	39
PERCENTAGE IMPROVEMENT	0.00	54.00	21.38	0.00	0.0
HEURISTIC TWO SOLN	605.56	5.98	22.66	791	39
PERCENTAGE IMPROVEMENT	0.0	50.63	19.71	0.0	0.0
HEURISTIC THREE SOLN	605.86	4.54	20.82	794	39
PERCENTAGE IMPROVEMENT	0.0	61.5	25.80	-0.37	0.0

113082

TABLE 7.2.3

SUMMARY OF RESULTS

PROBLEM NO 3

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	9793.73	21.88	58.44	801.0	31.0
HEURISTIC ONE SOLN	8055.33	17.13	44.38	801.0	29.0
PERCENTAGE IMPROVEMENT	17.75	21.72	24.0	0.0	6.45
HEURISTIC TWO SOLN	9793.73	21.88	52.47	807.0	31.0
PERCENTAGE IMPROVEMENT	0.0	0.0	10.21	-0.74	0.0
HEURISTIC THREE SOLN	8055.33	17.13	49.91	801	29
PERCENTAGE IMPROVEMENT	17.75	21.72	14.59	0.0	6.45

TABLE 7.2.4

SUMMARY OF RESULTS

PROBLEM NO 4 .

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	21978.87	92.88	220.67	1085.0	36.0
HEURISTIC ONE SOLN	19976.33	52.01	144.52	1085.0	36.0
PERCENTAGE IMPROVEMENT	9.11	56.93	22.04	0.0	0.0
HEURISTIC TWO SOLN	21759.8	49.83	145.95	1125.0	36.0
PERCENTAGE IMPROVEMENT	1.12	46.73	21.29	-3.6	0.0
HEURISTIC THREE SOLN	19976.7	43.6	143.06	1142.0	36.0
PERCENTAGE IMPROVEMENT	9.11	46.57	22.85	5.25	0.0

TABLE 7.2.5

SUMMARY OF RESULTS

PROBLEM NO 5

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	11104.33	85.95	280.83	758	28
HEURISTIC ONE SOLN	10292.84	47.58	223.02	758	28
PERCENTAGE IMPROVEMENT	7.37	55.43	20.58	0.0	0.0
HEURISTIC TWO SOLN	11098.40	49.32	246.83	777.0	28
PERCENTAGE IMPROVEMENT	0.2	42.62	12.11	0.0	0.0
HEURISTIC THREE SOLN	10292.84	32.75	228.12	766	28
PERCENTAGE IMPROVEMENT	7.71	64.13	18.77	0.0	0.0

TABLE 7.2.6

SUMMARY OF RESULTS

PROBLEM NO 6

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	17529.48	33.74	76.50	1457	66
HEURISTIC ONE SOLN	16980.52	18.45	61.21	1457	68
PERCENTAGE IMPROVEMENT	3.13	45.3	19.98	0.0	3.03
HEURISTIC TWO SOLN	17529.48	33.74	76.69	1462	69
PERCENTAGE IMPROVEMENT	0.0	0.0	-0.26	-0.34	4.55
HEURISTIC THREE SOLN	16980.52	18.45	61.21	1457	70
PERCENTAGE IMPROVEMENT	3.13	45.3	19.98	0	6.06

TABLE 7.2.7

SUMMARY OF RESULTS

PROBLEM NO 7

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	6293.62	72.48	129.15	716	12
HEURISTIC ONE SOLN	6185	30.72	87.39	716	12
PERCENTAGE IMPROVEMENT	1.72	57.62	32.33	0.0	0.0
HEURISTIC TWO SOLN	6293.62	7.86	66.86	724	12
PERCENTAGE IMPROVEMENT	0.0	73.46	48.23	-1.11	0.0
HEURISTIC THREE SOLN	6185	26.68	84.02	718	12
PERCENTAGE IMPROVEMENT	1.72	63.19	65.04	-0.28	0.0

TABLE 7.2.8

SUMMARY OF RESULTS

PROBLEM NO 8

	AGGREGATE LATENESS	ACTUAL INVENTORY COST ('000)	TOTAL COST ('000)	NO OF SET UPS	NO OF STOCK OUTS
STARTING SOLUTION	4756.10	21.83	89.35	482	18
HEURISTIC ONE SOLN	4493.10	8.57	76.09	482	19
PERCENTAGE IMPROVEMENT	5.53	60.74	14.84	0.0	5.35
HEURISTIC TWO SOLN	4756.10	13.46	80.94	498	18
PERCENTAGE IMPROVEMENT	0.0	38.34	9.41	3.33	0.0
HEURISTIC THREE SOLN	4493.10	5.22	70.77	4.83	18
PERCENTAGE IMPROVEMENT	5.53	76.12	20.79	0.27	0.0

TABLE 7.2.9

IMPROVEMENT IN AGGREGATE LATENESS

PROBLEM NO.	%IMPROVE-MENT OF H1	%IMPROVE-MENT OF H2	%IMPROVE-MENT OF H3
1	8.32	5.77	8.32
2	0.0	0.0	0.0
3	17.75	0.0	17.75
4	9.11	1.1	9.11
5	7.31	0.2	0.00
6	3.13	0.0	3.13
7	1.72	0.0	1.72
8	5.53	0.0	5.53
MEAN	6.61	0.88	5.70
S.D.	5.54	2.01	5.99

TABLE 7.2.10

IMPROVEMENT IN ACTUAL INVENTORY CARRYING COST

PROBLEM NO.	%IMPROVE-MENT OF H1	%IMPROVE-MENT OF H2	%IMPROVE-MENT OF H3
1	39.54	23.37	58.30
2	54.00	50.63	61.52
3	24.00	0.00	21.72
4	56.93	46.73	46.57
5	55.43	42.62	64.13
6	45.30	.0	45.30
7	52.62	73.46	63.19
8	60.44	38.34	76.08
MEAN	48.53	34.39	54.60
S.D.	11.95	25.39	16.56

TABLE 7.2.11

IMPROVEMENT IN NUMBER OF STOCK OUTS

PROBLEM NO.	%IMPROVE-MENT OF H1	%IMPROVE-MENT OF H2	%IMPROVE-MENT OF H3
1	4.02	3.0	4.02
2	0.0	0.0	0.0
3	6.45	0.0	6.45
4	0.0	0.0	0.0
5	0.00	0.00	0.00
6	3.03	4.55	6.06
7	0.0	0.0	0.0
8	5.56	0.00	0.0
MEAN	3.57	0.94	3.58
S.D.	2.59	1.79	3.05

TABLE 7.2.12

IMPROVEMENT IN TOTAL COST(THEORETICAL)

PROBLEM NO.	%IMPROVE-MENT OF H1	%IMPROVE-ENT OF H2	%IMPROVE-MENT OF H3
1	11.88	6.48	16.76
2	21.38	19.71	25.80
3	24.00	67.43	72.46
4	22.04	21.29	22.85
5	20.58	12.11	18.77
6	19.98	-0.26	19.98
7	32.33	48.23	65.04
8	14.84	9.414	20.79
MEAN	20.88	23.05	32.81
S.D.	6.10	23.11	22.43

A summary of results of the simulation carried out with eight different problems are given in the tables 7.2.1 to 7.2.8. These tables contain the starting solution, the improved solutions of these heuristics with the percentage improvement. Simple mean and standard deviation of the percentage improvement of every heuristic on all performance measures considered was calculated and tabled in 7.2.9 to 7.2.12.

7.3. Interpretation of Results

The results obtained in this study gives a surprise. Particularly, heuristic 1 improves the overall performance in a significant way. The standard deviation of the improvement of heuristic 1 and 3 seem to be too much , but, the fact is that when some jobs get delayed and some jobs get expedited, it is very difficult to assess the dynamic impact on " shop dynamics ".

On an average, the heuristic 1 improved the aggregate late ness by 6.61%, actual inventory costs by 48.53 %, number of stock outs by 3.57 % and total cost by 20.68%. The heuristic 2 doesn't improve the aggregate lateness in a significant way. But, on an average it improved actual inventory carrying cost by 34.39%, number of stock outs by 0.94 %, aggregate lateness by 0.88 % and total cost by 23.05%. The results are justified by the fact that when lot sizes are small, the aggregate number of components waiting for other components will reduce which in turn reduce the actual inventory carrying costs. But, it may not be the same for aggregate lateness. Because, required amount of quantity may not be available for assembly.

The heuristic 3 improves the actual inventory carrying cost in a significant way, because it does the things namely, decrease in the lot size

and also changing the criticality. On an average this heuristic improved the actual inventory carrying cost by 54.60%. In the same time it increased the setup costs also. The improvement in the total cost using this heuristic is more than the increase in the total cost using heuristic 2. In aggregate lateness, the improvement is slightly less than that of heuristic 1, which implies that there is no need of changing the lot sizes to reduce the aggregate lateness.In number of stock outs, the average improvement is 3.58% which is equal to that of heuristic 1.

Since, heuristic 3 over performs heuristic 2 in all aspects except the number of stock outs, heuristic 3 is prefered to heuristic 1.

CHAPTER 8. CONCLUSION

Previous research in Material Requirement Planning has been concerned largely with the single pass lot sizing and also with different dispatching rules in a job shop environment. Little work has been done to address the uncoordinated arrival pattern of different components, which is a common problem in the multi-stage, multi-product production system. As already stated, coordination is vital since shortage of a single item can halt the assembly process, resulting in sharply diminishing productivity and increased inventory costs. It is indeed a significant achievement, if it is possible to reduce the work-in-process inventory by even small but substantial percentage.

With this in our mind, we developed a heuristic to schedule the arrival of sub-assemblies at an assembly centre in a more co-ordinated manner. Preliminary results indicate that such a heuristic improves the performance in a powerful manner. This heuristic could be refined in many ways to improve the performance of a dynamic system such as MRP.

REFERENCE

1. Baker,K.R., 1974, " Introduction to Sequencing and Scheduling ", John Wiley & Sons, New York.
2. Baker,C.T.,and Dzielinki,B.P., 1960, " Simulation of a Simplified Job Shop", Management Science, Vol. 6,311.
3. Bedworth & Baily, " Integrated Production Systems - Management, Analysis, Design ", John Wiley & Sons.
4. Biggs, J.R., 1979, " Heuristic lot sizing and sequencing Rules in a Multi-stage Production Inventory System", DecisionScience, Vol. 10, 96-115.
5. Biggs, J.R., 1985, " Priority Rules for Shop Floor Control in a Material Requirements Planning System Under Various Levels of Capacity", International Journal of Production Research, Vol. 23, No. 1, 33-46.
6. Billington,P.,McClain,J.,and Thomas,L.J., 1979, " The interaction of lead time determination, Lot Sizing Decisions and Capacity Planning in MRP Systems", Proceedings, American Institute of Decision Sciences, Eleventh Annual Meeting.
7. Blackburn.J.D., and Millan.R.A., " Improved Heuristics for Multi-stage Requirements Planning Systems ", 1982, Vol. 28, NO. 1, 44-56.
8. Blackstone,J.H.,Philips.D.T., and Hogg.G.L., 1982, " A State-of-the-Art survey of Dispatching Rules for Manufacturing Job Shop Operations ", International Journal of Production Research, Vol. 20, No. 1, 27-45.
9. Buffa, E.S., 1968, " Production - Inventory Systems: Planning and Control ", HomeWood, Illunios: Richard D. Irwin, Inc.

10. Collier,D,A., 1980, " A Comparison of MRP Lot Sizing Methods considering Capacity Change Costs ", Journal of Operation Management, August.

11. Grasso, E.T., and Taylor, B.W, 1984," A Simulation Based Experimental Investigation of Supply Timing Uncertainty in MRP Systems ", International Journal of Production Research ", Vol. 22, No. 3, 485-467.

12. Fry.T.D., Philipoon.P.R. and Markland.R.E., 1988, International Journal of Production Research, Vol. 26, No. 7, 1193-1223.

13. Goodwin.J.S. and Weeks.J.K., 1986, " Evaluating scheduling policies in a multi-level assembly system ", International Journal of Production Research, Vol. 24, No. 2, 247-257.

14. Gutzmann.K.M., and Wysk.R.A., 1986," Capacity Control Policies for MRP systems",International Journal of Production Research, Vol. 24, No. 2, 359-374.

15. Magad.E.L., and Amos.J.S. " Total Material Management The Frontier for Maximizing Profit in the 1990s ", Competitive Manufacturing Series, Van Nostrand Reinhold, New York.

16. Orlicky,J., 1975," Material Requirement Planning", McGraw-Hill Book Company, New York.

17. Panwalker.S.S. and Iskander.W., 1977, " A Survey of Scheduling Rules", Operation Research, Vol. 25, No. 1, January-february, 45-61.

18. Potty,V.S., 1990, " Simulation of Heuristic Lot Sizing and Sequencing Procedures in MRP ",Unpublished M.Tech.dissertation, Indian Institute of Technology, Kanpur.

19. Putnam,A.O., Everdell,R., Dornam,G.H., Cronen, R.R.,and Lindgren,L.H., 1971, "Updating Critical Ratio and Slack-time Priority Scheduling Rules", Production-Inventory Management, Vol. 12,51.

20. Russell, R.S., and Taylor, B.W., 1985, " An Evaluation of Sequencing Rules for an assembly shop", Decision Science,Vol. 16,196-212.

21. Sharma,R.R.K., and Potty,V.S., " Multi-pass Heuristic for Improved Stockout Performance", 1991, Design Automation and Computer Integrated Manufacturing, Tata-McGrewHill, NewDelhi.

22. South, J.B., and Steward, R.J., 1986, " A Increase in Lot Size and Planned Manufactured Lead Time with No Increase in Work-in-process Inventory", Production and Inventory Management , Vol. 27, No. 1, 19-25.

23. Steele, D.C., 1973, The Nervous MRP system - Hoe to Battle ", Production & Inventory Management", Vol. 16, 81-85.

24. Vollman,T.E., Berry,W.L., and Whybark,D.C, 1989," Manufacturing Planning and Control Systems ", Galgotia Publications Pvt Ltd, New Delhi.

25. Wagner,H.M., and Whitin,T.M., 1958, " Dynamic Version of Economic Lot Size Model", Management Science, October, 89-96.

APPENDIX A

SOFTWARE DEVELOPED

APPENDIX A

SOFTWARE DEVELOPED

```
program  mrp(input,output);

const
        max_mc_no                   = 15;
        max_process_no              = 5;
        max_assly_no                = 45;
        max_child_assly             = 5;
        max_period_no               = 40;
        infinity                    = 32000;
        max_statistics_pt_no        = 4;

type
        max_assly_mc_no = 16..20;

        event_type   =   (collect_statistics,
                          processing_gets_over,
                          assly_gets_over);

        ee = record
                no,
                early_demand    :integer;
                waiting_time    : real;
                qty_needed      :array [1..max_period_no] of integer;
                arrival_time    :array [1..max_period_no] of real;
                qty_arrived     :array [1..max_period_no] of integer;
                lot_number      :array [1..max_period_no] of integer;
             end;

        ptr_to_lot  =  ^lot;

        lot   = record
                  assly_no,
                  size,
                  current_mc_no     : integer;
                  operation_number  :integer;
                  total_wait_time,
                  work_rem,
                  due_time,
```

```
            crr_value,
            pt,
            time_of_joining_the_queue  :  real;
            next  :  ptr_to_lot;
          end;

ptr_to_assly  =  ^ lot2;

lot2  =  record
            assly_no,
            current_mc_no,
            early_assly_order,
            order_completed,
            no_of_childs    :integer;
            late_ness       :real;
            child_infor     :array[1..max_child_assly]  of  ee;
            next            : ptr_to_assly;
          end;

process_data  =  record
                    mc_no  :  integer;

                    setup_time,
                    processing_time_per_unit,
                    setup_cost  :  real
                  end;

child_assly  =  record
                     child_assly_no,
                     qty_to_be_contributed  :  integer;
                   end;

assly  =  record
            assly_no,
            transformed_assly_no,
            parent_no,
            qty_for_parent,
            no_of_child_asslys,
```

```
                no_of_machines_needed,
                mc_no_for_assly,
                no_of_processing_needed    : integer;

                cost,assembly_time:   real;

                needs_processing   :  boolean;

                array_of_processing_sequence
                        : array[1..max_process_no]  of  process_data;

                array_of_child_asslys
                        :    array[1..max_child_assly]
                                of   child_assly;
            end;
processing_machine = record
                        work_in_progress  :  boolean;
                        utilisation_time,
                        finish_time  :  real;
                        no_of_jobs_waiting   :integer;
                        lot_ptr,
                        ptr_to_lot_being_processed   :   ptr_to_lot;
                      end;

assly_machine = record

                    assly_no_being_assembled,
                    current_lot_size     :integer;
                    work_in_progress  :  boolean;
                    finish_time    :real;
                    assly_ptr    :ptr_to_assly;
                  end;



(* newly declared types *)

data_for_period = record
```

```
                    demand1  :integer;
                    setup_cost,
                    inv_carry_cost  :  real;
                  end;


    tables = record
          scheduled_rec,
          projected_on_hand,
          gross_req,
          net_req,
          lot_quantities,
          planned_order    :integer;
          ordering_cost,
          inv_carry_cost   :real;
          end;

    inv_record =record
           qty_on_hand,
           open_reple_order,
           lead_time   :integer;
           cost:real;
           aa  :array[1..max_period_no]  of  tables;
            end;
    event_record = record
                     event  :  event_type;
                     relevant_no  :  integer;
                     time  :  real;
                   end;

var
      clock_time,
      simulation_time,
      waiting_time_during_processing,
      initial_time,
      ordi_size,
      set_up_cost,
      inven_cost,
```

```
        total_cost,
        next_event_at,
        umi,
        end_late,
        best_late,
        cost            : real;
        continue,iterations_can_continue  :  boolean;
        temp_record  :data_for_period;
        ooo,kk,mn,mmm,g,f,h,o,oo:text;
        input_data_for_ww :array[1..max_period_no] of data_for_period;
        iteration_required,best_iteration,no_of_stockout,no_of_setup,
        heuristic,stock_out,option,iteration_no,
        aaa,yy,current_period,i,no_of_period,mm,lot_no     :integer;
        array_for_basic_data_about_asslys  :array[1..max_assly_no] of assly;
        demand  :array[1..max_assly_no,1..max_period_no] of integer;
        array_for_process_mc:array[1..max_mc_no] of processing_machine;
        array_for_assly_mc :array[max_assly_mc_no] of assly_machine;
        item,base   :ptr_to_lot;
        succ,pred   :ptr_to_assly;
        inv_array  :array[1..max_assly_no] of inv_record;
        priority    :array[1..max_assly_no] of integer;
        late_data  :array[1..max_assly_no] of real ;
        aggre_late  :array[1..20] of real;
        inv_status  :array[1..max_assly_no] of integer;
        k,past_history:array[1..max_assly_no]  of  integer;
        diff,there_cost,actual_cost  :array[1..max_assly_no]  of  real  ;
        r,s  :set  of  0..max_assly_no;

procedure   read_assly_data;

var
        ff     :text;
        assembly_no,
        child_assly_no,
        needs_process,
        no,product_no,
        process_no          :integer;
```

```
begin (* read_assly_data *)
        assign(ff,'assly8.dat');
        reset(ff);
        assembly_no := 0;
        while not eof(ff) do
        begin
            assembly_no :=assembly_no +1;
            writeln(assembly_no);
            readln(ff,
                array_for_basic_data_about_asslys[assembly_no].assly_no,
                  array_for_basic_data_about_asslys[assembly_no].
                        no_of_child_asslys,
                needs_process,
                 array_for_basic_data_about_asslys[assembly_no].cost);
          writeln(array_for_basic_data_about_asslys[assembly_no].assly_no);
          array_for_basic_data_about_asslys[assembly_no].
                needs_processing := (needs_process = 1);

          if    (array_for_basic_data_about_asslys[assembly_no].
                        no_of_child_asslys > 0 )
          then
                begin
                for i :=1 to array_for_basic_data_about_asslys[assembly_no].
                                no_of_child_asslys do
                with array_for_basic_data_about_asslys[assembly_no] do
                with array_of_child_asslys[i] do
                readln(ff,child_assly_no,qty_to_be_contributed);
                end;

                if    array_for_basic_data_about_asslys[assembly_no].
                   no_of_child_asslys > 1 then
                begin
                readln(ff,array_for_basic_data_about_asslys[assembly_no].
                mc_no_for_assly,
                          array_for_basic_data_about_asslys[assembly_no].
                          assembly_time);
                end;
```

```
            if    (array_for_basic_data_about_asslys[assembly_no].
                                              needs_processing)
            then
              begin
                with  array_for_basic_data_about_asslys[assembly_no]  do
                begin
                readln(ff,transformed_assly_no,
                                           no_of_processing_needed);
                parent_no := 1;
                end;
                for  process_no := 1 to
                   array_for_basic_data_about_asslys[assembly_no].
                                 no_of_processing_needed  do
                with  array_for_basic_data_about_asslys[assembly_no]  do
                with  array_of_processing_sequence[process_no]  do
                begin
                readln(ff,mc_no,setup_time,
                          processing_time_per_unit,setup_cost);
                end;
              end

          else  (*  collect_parent_data  *)
                with  array_for_basic_data_about_asslys[assembly_no]  do
                readln(ff,parent_no,qty_for_parent);
           end;

close(ff);

end;  (*  read_assly_data  *)

procedure read_assly_demands;

var
        input_demand_data   :text;

        assly_no,
        aaa,
```

```
        period_no,
        no_of_demands,
        demand_no              :integer;

begin (* read_assly_demands  *)

        for assly_no := 1 to max_assly_no do
        for period_no := 1 to max_period_no do
        demand[assly_no,period_no] := 0;

        assign(input_demand_data,'demand8.dat');
        reset(input_demand_data);

        while not eof (input_demand_data) do
        begin
          readln(input_demand_data,assly_no,no_of_demands);
          writeln(assly_no);
          for demand_no := 1 to no_of_demands do
          begin
            read(input_demand_data,period_no);
            readln(input_demand_data,demand[assly_no,period_no]);
         end;
        end;

        (* writeln('   FROM DEMAND DATA FILE  ');*)

        for assly_no := 1 to max_assly_no do
        for period_no := 1 to max_period_no do
        inv_array[assly_no].aa[period_no].gross_req     :=
                                              demand[assly_no,period_no];

close(input_demand_data);

end; (* read_assly_demands *)

procedure  read_input_data;

begin (* read_input_data *)
```

```
        read_assly_data;
        read_assly_demands;

end; (* read_input_data *)

procedure    initialisation;
var
ii,nnn:integer;
inv_data:text;

begin
  assign(inv_data,'inv8.dat');
  reset(inv_data);
  for i := 1 to max_assly_no do
  with  inv_array[i]  do
  begin
  writeln(i);
  (*    readln(inv_data,qty_on_hand,open_reple_order,cost,lead_time);    *)
  readln(inv_data,nnn,lead_time);
  writeln(nnn);
  open_reple_order := 0 ;
  qty_on_hand := 0;
  for ii := 1 to max_period_no do
  with  aa[ii]  do
  begin
  read(inv_data,ordering_cost);
  ordering_cost := ordering_cost * 5 ;
  (*  if  array_for_basic_data_about_asslys[i].no_of_child_asslys  >  1  then
  inv_carry_cost := 3.0 else inv_carry_cost := 1.0; *)
  scheduled_rec := 0;
  projected_on_hand := 0;
  net_req := 0;
  lot_quantities  :=  0;
  planned_order  := 0;
  end;
  for ii := 1 to max_period_no do
  with  aa[ii]  do
```

```
  begin
  read(inv_data,inv_carry_cost);
  inv_carry_cost  :=  inv_carry_cost;
  end;
  end;
  for i:= 1 to max_assly_no do
  inv_array[i].aa[1].scheduled_rec    :=    inv_array[i].open_reple_order;
close(inv_data);
end;

procedure     do_lot_sizing(n:integer);

var
  optimum_policy  :array[1..max_period_no]  of  integer;
  temp_array  :array[1..max_period_no,1..max_period_no]  of  real  ;
  t_star,sum,i,j,t,l,h,k,x,ii:integer;
  sum1,min1,min2   :real;
  ff,order   :array[1..max_period_no]of   real;

begin  (*  do_lot_sizing  *)

{*   initialisation   *}

 for i := 1 to max_period_no do
   with  input_data_for_ww[i]  do
    begin
        demand1:=     inv_array[n].aa[i].net_req;
        setup_cost    :=    inv_array[n].aa[i].ordering_cost;
        inv_carry_cost    :=    inv_array[n].aa[i].inv_carry_cost;
    end;

(* for  i  :=  1  to  max_period_no  do  writeln(input_data_for_ww[i].inv_carry_cost:

       t_star  :=  1;
       j  :=  1;
       l  :=  0;

for i := 1 to max_period_no do
```

```
    begin
    temp_array[i,1]  :=  0.0;
    temp_array[i,2]   :=0.0;
    end;

for current_period := 1 to max_period_no do
begin
  with input_data_for_ww[current_period] do
  begin
    l := 0;
    for j :=1 to current_period-1 do
    begin
      suml :=0;
      if((current_period-j)*inv_carry_cost * demandl < setup_cost ) then
      begin
        (* summation over h & k *)

        for h := j to current_period -1 do
        for k := h+1 to current_period do
        suml := suml +input_data_for_ww[h].inv_carry_cost
                    * input_data_for_ww[k].demandl ;

        (* STORING IN A ARRAY *)
        l  :=  l+1;
        if  j  <>1 then temp_array[1,1] := suml+input_data_for_ww[j].setup_cost
                                                              +  ff[j-1]
                    else temp_array[1,1] := suml
                                          +input_data_for_ww[j].setup_cost   ;
        temp_array[1,2] := j ;
      end;
    end; (* for j< current period loop ends *)

    {* calculation for the current period t *}

    l  :=  l+1;
    if  current_period = 1 then temp_array[1,1] := setup_cost
                           else temp_array[1,1] := setup_cost +
                           ff[current_period-1];
```

```
    temp_array[1,2] := current_period;
  end; (* WITH LOOP ENDS *)
  {* calculation of the minimum in temp_array *}
  min1 := temp_array[1,1];
  min2 := temp_array[1,2];
  for i := 1 to l do
  begin
    if temp_array[i,1] < min1 then
    begin
      min1 :=temp_array[i,1];
      min2 := temp_array[i,2];
    end;
  end;

  ff[current_period] := min1;
  t_star := round(min2) ;
  optimum_policy[current_period] := t_star;
end; (* FOR ENDS *)

{* finding the optimal Orders *}
for i := 1 to max_period_no do  order[i] := 0;
i := max_period_no;
total_cost := total_cost + ff[max_period_no];
there_cost[n] := ff[max_period_no];
(*  writeln(ooo,n:5,'  theretical  cost',there_cost[n]:8:2);  *)
min1 := there_cost[n];
repeat
  x := optimum_policy[i];
  sum :=0;
  for ii := round(x) to i do sum := sum +input_data_for_ww[ii].demand1;
  order[x] :=sum;
  min1 := min1 - input_data_for_ww[x].setup_cost;
  inv_array[n].aa[x].lot_quantities := sum;
  i :=x-1;
until x =1;
inven_cost := inven_cost + min1 ;
set_up_cost := set_up_cost + ( there_cost[n] - min1 );
(*   writeln(ooo,'setup cost ',there_cost[n] -min1:8:2);  *)
```

```
 end;  (* LOT SIZING ENDS *)

procedure     explore(assly_no:integer);
var
j,cc:integer;
begin
     with  array_for_basic_data_about_asslys[assly_no]  do
     if  no_of_child_asslys > 0  then
     begin
       for  j  :=  1  to  no_of_child_asslys  do
       begin
       with  array_of_child_asslys[j]  do
       begin
       cc:=array_for_basic_data_about_asslys[assly_no].
       array_of_child_asslys[j].child_assly_no;
       for  i  :=  1  to  max_period_no  do
       inv_array[cc].aa[i].gross_req     :=     inv_array[assly_no].aa[i].gross_req
                              *     array_of_child_asslys[j].qty_to_be_contributed
                              +     inv_array[cc].aa[i].gross_req;

       array_for_basic_data_about_asslys[assly_no].parent_no   :=   -1;
       array_for_basic_data_about_asslys[cc].parent_no    :=
       array_for_basic_data_about_asslys[cc].parent_no  -  1  ;
       end;{  with}
       end;
       end
       else   array_for_basic_data_about_asslys[assly_no].parent_no   :=   -1;
 end;

procedure     do_four_basic_steps(assly_no:integer);

var
j,cc,pp,order_time:integer;
begin
     (*     writeln(f,assly_no);
     write(f,'TimePeriod    current');
     for  j  :=  1  to  max_period_no  do  write(f,'  ',j:4);
```

```
writeln(f);
write(f,'external dem        ');
for j := 1 to max_period_no do write(f,' ',demand[assly_no,j]:4);
writeln(f);

write(f,'gross               ');
for j := 1 to max_period_no do write(f,' ',
                inv_array[assly_no].aa[j].gross_req:4);
writeln(f);   *)

(* netting *)

with inv_array[assly_no] do
begin

(* write(f,'schd rec            ');
for i := 1 to max_period_no do write(f,' ',aa[i].scheduled_rec:4);
writeln(f);   *)
for i := 1 to max_period_no do
with aa[i] do
begin
case i of
1 :begin
    pp := qty_on_hand + scheduled_rec -
    inv_array[assly_no].aa[i].gross_req    ;
    if pp > 0 then
           begin
             projected_on_hand := pp;
             net_req := 0;
           end
          else
           begin
              projected_on_hand := abs(pp);
             net_req := abs(pp);
           end;
   end;
2..max_period_no : begin
                   if aa[i-1].net_req = 0 then pp :=
```

```
                        aa[i-1].projected_on_hand  +  scheduled_rec
                            -     inv_array[assly_no].aa[i].gross_req
                         else pp := scheduled_rec -
                         inv_array[assly_no].aa[i].gross_req;
                        if pp > 0 then
                  begin
                    projected_on_hand := pp;
                    net_req := 0;
                  end
                 else
                  begin
                    if aa[i-1].net_req = 0 then projected_on_hand := abs(pp)
                                           else projected_on_hand :=
                                           aa[i-1].projected_on_hand    +abs(pp);
                    net_req := abs(pp);
                  end;
            end;
      end; { case}
      end; { for i }
(*        write(f,'proj hand   ',qty_on_hand:5,'   ');
      for i := 1 to max_period_no do write(f,' ',aa[i].projected_on_hand:4);
      writeln(f);
      write(f,'net req              ');
      for i := 1 to max_period_no do write(f,' ',aa[i].net_req:4);
      writeln(f);    *)
     end; { with inv_rec }

  end; (* do _four_basic_steps *)

procedure   mrp_initialise;
var
assly_no,ii,order_time     :integer;

begin

repeat
  for assly_no := 1 to max_assly_no do
  with  array_for_basic_data_about_asslys[assly_no]  do
```

```
    begin
      writeln(assly_no:5,'     ',parent_no);
      if ( parent_no = 0 ) then explore(assly_no);
    end;
      ii := 1;
      for i := 1 to max_assly_no do
      begin
      if ( array_for_basic_data_about_asslys[i].parent_no <> -1 )
      then ii := 0 ;
      end
  until ( ii = 1 );
    set_up_cost := 0.0;
    inven_cost := 0.0;
    total_cost := 0.0;
    no_of_setup := 0;
    for assly_no := 1 to max_assly_no do
    begin
      writeln(f);
      writeln(' ASSEMBLY NO  ',assly_no);
      do_four_basic_steps(assly_no);
      do_lot_sizing(assly_no);
(*      write(f,'lot                    ');
      for i := 1 to max_period_no do write(f,' ',
      inv_array[assly_no].aa[i].lot_quantities:4);
      writeln(f);   *)
      for i := 1 to max_period_no do
      if  inv_array[assly_no].aa[i].lot_quantities  >  0  then
        begin
          order_time := i - inv_array[assly_no].lead_time;
          if order_time < 1 then order_time := 1 ;
          inv_array[assly_no].aa[order_time].planned_order    :=
          inv_array[assly_no].aa[i].lot_quantities      +
          inv_array[assly_no].aa[order_time].planned_order    ;
        end;

      write(f,'planned order        ');
      for i := 1 to max_period_no do write(f,' ',
      inv_array[assly_no].aa[i].planned_order:4);
```

```
      writeln(f);

      writeln(f,'lead time                    ',inv_array[assly_no].lead_time:4);
    end;
    for assly_no := 1 to max_assly_no do
    for i := 1 to max_period_no do
    if  inv_array[assly_no].aa[i].planned_order  >  0
    then no_of_setup := no_of_setup + 1 ;
  end; { mrp_inio }

procedure    insert_in_assly_mc(item2:ptr_to_assly);
var
  pred_ptr,succ_ptr   :ptr_to_assly;

begin
  with array_for_assly_mc[item2^.current_mc_no] do
  begin
    if assly_ptr = nil then begin
                              assly_ptr := item2;
                            end
       else
       begin
       pred_ptr := assly_ptr;
       succ_ptr := assly_ptr^.next;
       while succ_ptr <> nil do
             begin
             pred_ptr := succ_ptr;
             succ_ptr := succ_ptr^.next;
             end;
       pred_ptr^.next := item2;
       end; { else }
    pred_ptr  :=  array_for_assly_mc[item2^.current_mc_no].assly_ptr;
    while(not(pred_ptr = nil)) do
       begin
            pred_ptr := pred_ptr^.next;
       end; { while }
    end;   { with }
end;    { insert over }
```

```
procedure     load_the_mc(ij:integer;time:real);
var
pred_ptr,least,succ_ptr     :ptr_to_lot;
min_crr:real;
pred_ptr1,succ_ptr1     :ptr_to_assly;

begin
case ij of
1..max_mc_no :
  with array_for_process_mc[ij] do

  if (( lot_ptr <> nil ) and ( work_in_progress = false )
        and ( no_of_jobs_waiting > 0 )
        and ( ptr_to_lot_being_processed = nil )) then
  begin
    mm:=mm+1;

    pred_ptr  :=  lot_ptr;
    succ_ptr  :=  nil;
    least     :=  nil;
    min_crr   :=  100000.00;

    while pred_ptr <> nil do
    begin
      pred_ptr^.crr_value  :=
      (( pred_ptr^.due_time - time) - pred_ptr^.work_rem ) /
      pred_ptr^.work_rem ;

      if pred_ptr^.crr_value < min_crr then

      begin
        array_for_process_mc[ij].ptr_to_lot_being_processed  :=  pred_ptr;
        least                              := succ_ptr;
        min_crr                            := pred_ptr^.crr_value;
      end;
      succ_ptr := pred_ptr;
```

```
        pred_ptr := pred_ptr^.next;

    end;

    if ( least = nil ) then
                        lot_ptr := lot_ptr^.next
                        else least^.next := ptr_to_lot_being_processed^.next;

    array_for_process_mc[ij].ptr_to_lot_being_processed^.next   :=   nil;

    work_in_progress:= true;
    no_of_jobs_waiting := no_of_jobs_waiting - 1;

    with    array_for_basic_data_about_asslys[ptr_to_lot_being_processed^.
                                                    assly_no] do
    with    array_of_processing_sequence[ptr_to_lot_being_processed^.
    operation_number] do
    array_for_process_mc[ij].finish_time := time +
    ptr_to_lot_being_processed^.pt;

    if finish_time > simulation_time then
                utilisation_time := utilisation_time +
                ( simulation_time -time )
                else utilisation_time := utilisation_time +
                ptr_to_lot_being_processed^.pt;

    with ptr_to_lot_being_processed^ do
    total_wait_time := total_wait_time +
    ( time - time_of_joining_the_queue );

    base := array_for_process_mc[ij].lot_ptr;
    while base <> nil do
    begin
          base := base^.next;
    end;
  end;

16..20  :;
```

```
end; { case }
end; { for all assl }

procedure    insert(itemme:ptr_to_lot);
var
pred_ptr1,succ_ptr1    :ptr_to_lot;
ii:integer;

begin
    with array_for_process_mc[itemme^.current_mc_no] do
      begin
      base := lot_ptr;
      if lot_ptr = nil then begin
                              lot_ptr := itemme;
                            end
        else
        begin
        pred_ptr1 := lot_ptr;
        succ_ptr1 := lot_ptr^.next;
        while succ_ptr1 <> nil do
              begin
              pred_ptr1 := succ_ptr1;
              succ_ptr1 := succ_ptr1^.next;
              end;
        pred_ptr1^.next := itemme;
        end; { else }

  no_of_jobs_waiting:=no_of_jobs_waiting + 1;

  pred_ptr1 := array_for_process_mc[itemme^.current_mc_no].lot_ptr;
  while(not(pred_ptr1 = nil)) do
     begin
          pred_ptr1 := pred_ptr1^.next;
     end; { while }

     end;     { with}
```

```
end; {insert }

procedure    load_the_assly_mc(timee:real);
var
aa,ii,j,m     :integer;
w_t  :real;
load,ini_found     :boolean;
pred_ptr   :ptr_to_assly;
wait_time  :array [1..max_child_assly] of real ;

begin

  for ii := 1 to max_child_assly do
  wait_time[ii]  :=  0.0;

  for ii := 16 to 20 do
  with  array_for_assly_mc[ii] do

  begin
    (*  writeln('ass  mc  no  ',ii);  *)
    pred_ptr  :=  assly_ptr;
    j:=0;
    while pred_ptr <> nil do

    begin
      with pred_ptr^ do

      begin
         load  :=  true ;
         for  j  :=  1  to  no_of_childs do

         begin
           if   early_assly_order   >=   child_infor[j].early_demand
                then  load  :=  false;
         end;

         case load of
         true  :  begin
```

```
if current_period > early_assly_order then
no_of_stockout := no_of_stockout + 1;
with array_for_assly_mc[pred_ptr^.current_mc_no] do
begin
  current_lot_size  :=
  inv_array[pred_ptr^.assly_no].aa[early_assly_order].
  planned_order;
  work_in_progress := true ;
  finish_time := timee +
  array_for_basic_data_about_asslys[pred_ptr^.assly_no].
  assembly_time;
  assly_no_being_assembled  :=  pred_ptr^.assly_no;

  if finish_time > early_assly_order * 4000.00 then
  late_ness := late_ness +
  (( finish_time - early_assly_order * 4000.00 ) *
  current_lot_size  ) ;

  order_completed := order_completed + 1 ;

  for j := 1 to no_of_childs do
  with  child_infor[j]  do

  begin
    m := 1 ;
    aa := 0;
    with   array_for_basic_data_about_asslys[pred_ptr^.
    assly_no]  do
    with  array_of_child_asslys[j]  do
    if  child_assly_no  =  child_infor[j].no  then
    begin
    aa  :=  array_of_child_asslys[j].qty_to_be_contributed  *
    inv_array[pred_ptr^.assly_no].aa[early_assly_order].
    planned_order;
    end;

    while  (  child_infor[j].qty_arrived[m]  <  1  )
    and ( m < 40 ) do m := m + 1;
```

```
    repeat
    if  child_infor[j].qty_arrived[m] > 0 then
    begin
    if  child_infor[j].qty_arrived[m] <= aa then
    begin
      wait_time[j] := qty_arrived[m]
        * ( timee - arrival_time[m] ) + wait_time[j];
      aa := aa - qty_arrived[m];
      qty_arrived[m]  := -1 ;
    end

    else
      begin
      wait_time[j] := aa
        * ( timee - arrival_time[m] ) + wait_time[j];
      qty_arrived[m] := qty_arrived[m] - aa ;
      aa := 0;
    end;

    end;
    m:= m +1;
    until ( aa = 0 ) or ( m = 41 );

    wait_time[j] := wait_time[j] / 4000.00;
end; { for j }

for j := 1 to no_of_childs do
begin

  aa := 0;
  w_t := 0.0;
   with   array_for_basic_data_about_asslys[pred_ptr^.assly_n
   with  array_of_child_asslys[j]  do
   if  child_assly_no  =  child_infor[j].no  then
   begin
   aa  :=  array_of_child_asslys[j].qty_to_be_contributed  *
   inv_array[pred_ptr^.assly_no].aa[early_assly_order].plan
```

```
                    end;

                    inv_status[child_infor[j].no]      :=
                    inv_status[child_infor[j].no]  -  aa  ;

                    w_t := aa * wait_time[j] *
                     inv_array[child_infor[j].no].aa[current_period].inv_car
                    w_t := w_t / current_lot_size ;
                    child_infor[j].waiting_time     :=     child_infor[j].waiting_ti
                    end;

                  child_infor[j].qty_needed[early_assly_order]    :=    -1;

                  j:=early_assly_order  +  1;
                  ini_found  :=  false  ;
                  repeat
                    if  (  inv_array[assly_no].aa[j].planned_order  >  0  )  then

                    begin
                       early_assly_order  :=  j  ;
                       ini_found  :=  true  ;
                    end;

                    j  :=  j  +1;

                 until  ((  ini_found  =  true  )  or  (  j  =  max_period_no  +  1));

                 if  (  j  =  max_period_no  +  1  )  then
                                          begin
                                          early_assly_order  :=  1;
                                          end;


               end;
             end;

    end;  {  case  }

end;  {  pred_ptr  }
```

```
      pred_ptr  :=  pred_ptr^.next;

    end; { pred <> nil }

  end; { with array for all mc }

end; { load the assly mc }

procedure      search_for_parent(tt,amt:integer;times:real);
var
np,nq,nn,ii,j,jj,m:integer;
pred_ptr,succ_ptr    :ptr_to_assly;
over,load,found,ini_found   :   boolean;

begin
lot_no  :=  lot_no  +  1 ;
case   array_for_basic_data_about_asslys[tt].parent_no   of
0  :begin
      inv_status[tt]  :=  inv_status[tt]  -  amt  ;
    end;

1   :begin

      nn  :=  0;
      nq  :=  0;

   ii   :=   16;
   found   :=   false;
   over   :=   false  ;
   while  (  ii  <=  20  )  and  (  found  =  false  )  do

   begin
     with   array_for_assly_mc[ii]   do

     begin
       pred_ptr   :=   assly_ptr;
       j:=0;
       while  (  pred_ptr  <>  nil  )  and  (  found  =  false  )  do
```

```
begin

  with pred_ptr^ do

  begin
    for j := 1 to no_of_childs do

    begin
      if child_infor[j].no = tt then

      begin
        found := true ;
        jj := child_infor[j].early_demand;
        child_infor[j].qty_arrived[current_period] := amt;

        repeat

        jj := child_infor[j].early_demand;

        if ( amt >= child_infor[j].qty_needed[jj] ) and
           ( child_infor[j].qty_needed[jj] > 0 ) then

        begin

          amt := amt - child_infor[j].qty_needed[jj] ;
          child_infor[j].qty_needed[jj] := 0;
          writeln(h,'fully      satisfied');

          m := jj +1 ;
          ini_found := false ;
          repeat
             if ( child_infor[j].qty_needed[m] > 0 ) then

             begin
               child_infor[j].early_demand := m ;
               ini_found := true ;
             end;
```

```
                    m := m +1;

                until (( ini_found = true ) or ( m = max_period_no + 1 ));

              end

              else
                 begin

                   child_infor[j].qty_needed[jj]     :=
                   child_infor[j].qty_needed[jj]  -  amt  ;
                   amt := 0;
                 end;


              until ( amt = 0 ) or ( m = max_period_no ) or ( over = true );

              child_infor[j].arrival_time[current_period]    :=    times;
              child_infor[j].lot_number[current_period]  :=  lot_no  ;
            end; { if child found }

          end; { for all childs }

        end; { with pred }

        pred_ptr  :=  pred_ptr^.next;

      end; { while pred <> nil }

    end; { with mc ii }

    ii  :=  ii  +  1  ;

  end; { while ii < 21 }

end; { case parent =1 }

2..10   :
```

```
begin

while ( amt > 0 ) do
begin

  nn := max_period_no;
  nq := 0;

  for ii := 16 to 20 do
  with  array_for_assly_mc[ii] do

  begin
    (*  writeln('ass  mc  no  ',ii);*)
    pred_ptr  :=  assly_ptr;
    j:=0;
    while pred_ptr <> nil do

    begin
      with pred_ptr^ do

      begin
        (*      writeln('assly_no',assly_no);*)
        for  j  :=  1  to  no_of_childs  do
        begin
          (*      writeln('child      ',child_infor[j].no:3);*)
          if   child_infor[j].no  =  tt  then

          begin
            (*          writeln(h,'child',child_infor[j].no,
          'early',child_infor[j].early_demand,
          'qty      needed      ',child_infor[j].qty_needed[child_infor[j].
          early_demand]);*)

          if  child_infor[j].early_demand  <  nn  then
          begin
            nn   :=   child_infor[j].early_demand;
            nq   :=    child_infor[j].qty_needed[nn];
          end
```

```
                else
                if ( child_infor[j].early_demand = nn ) and
                   (       child_infor[j].qty_needed[child_infor[j].early_demand]
                   < nq ) then
                     nq  :=  child_infor[j].qty_needed[nn]  ;
              end; { if child found }

          end; { all child }

      end; { with prd ptr }

      pred_ptr  :=  pred_ptr^.next;

    end; { while pred <> nil }

end; { for all mc }

{ early demand for all parents found }

{ going inside and satisfying early demand }

writeln(h,'after  searh  nn  ',nn:4,'nq  ',nq:4);

    if nq <> 0 then

    for ii := 16 to 20 do
    with  array_for_assly_mc[ii]  do

    begin
      (*  writeln('ass  mc  no  ',ii);  *)
      pred_ptr  :=  assly_ptr;
      j:=0;
      while pred_ptr <> nil do

      begin
        with pred_ptr^ do

        begin
```

```
(*      writeln('assly_no',assly_no);*)
for j := 1 to no_of_childs do

begin
  (*      writeln('child      ',child_infor[j].no:3);*)
  if ( ( child_infor[j].no = tt ) and
       ( child_infor[j].early_demand = nn ) and
       ( child_infor[j].qty_needed[nn] = nq ) ) then

  begin
    jj  :=  child_infor[j].early_demand;
    (* writeln(h,'   parent no :',assly_no,' early period :',
    child_infor[j].early_demand,' qty needed ',
    child_infor[j].qty_needed[child_infor[j].early_demand]:5);        *)

    if ( amt < nq ) then

    begin
      child_infor[j].arrival_time[current_period]   :=   times;
      (*      writeln('c',child_infor[j].qty_needed[current_period]);     *)
      child_infor[j].qty_arrived[current_period]    :=  amt  +
        child_infor[j].qty_arrived[current_period];
      child_infor[j].qty_needed[nn]    :=    child_infor[j].qty_needed[nn]
         - amt ;
      child_infor[j].lot_number[current_period]   :=   lot_no;
      (*      writeln('cc',child_infor[j].qty_arrived[current_period]);     *)
      amt := 0;
    end
    else
    begin
      child_infor[j].arrival_time[current_period]   :=   times;
      child_infor[j].qty_arrived[current_period]    :=  nq  +
      child_infor[j].qty_arrived[current_period]      ;
      child_infor[j].lot_number[current_period]   :=  lot_no  ;
      child_infor[j].qty_needed[nn]     :=0;
      amt := amt - nq ;
      (*      writeln(h,'cc',child_infor[j].
      qty_arrived[current_period]);     *)
```

```
                m := nn +1 ;
                ini_found := false ;
                   repeat
                     if ( child_infor[j].qty_needed[m] > 0 ) then

                     begin
                       child_infor[j].early_demand := m ;
                       ini_found := true ;
                     end;

                     m := m +1;

                   until (( ini_found = true ) or ( m = max_period_no + 1 ));
              end;

              (* writeln(h, 'assly no  ',assly_no,'amt remaining  ',amt); *)
            end; (* if child is found *)

          end;
          { for all child }

        end; { with pred ptr }
        pred_ptr := pred_ptr^.next;

      end; { while pred ptr <> nil }

    end  { with mc }

    else   begin
             (* writeln(oo,'some problem '); *)
             amt := 0 ;       { take care of this }
           end;

     { most demanding person satisfied }

(* writeln(h,'amt after ',amt:4); *)

end; { while amt > 0 }
```

```
end   { if parent > 2 }

end; { case ends }

load_the_assly_mc(times);

end; { search for parent ends }


procedure        initialise_the_new_item(i,ss:integer;time:real);
var
 j,process_no,ii,yy      :integer;
 front:ptr_to_lot;
 item2:ptr_to_assly;

begin
  (* writeln(h,'   ASSEMBLY NO   initiated with   ',i, 'size ',ss:6);*)
  inv_status[i]  :=  inv_status[i]  +  ss  ;

  case   array_for_basic_data_about_asslys[i].needs_processing   of
  true   :
  begin
      new(item);
      with item^ do
      begin
        assly_no  :=  i;
        size := ss;              (* inv planned order was there *)
        total_wait_time    := 0.0;
        crr_value          := 0.0;
        if  heuristic  =  1  then
        due_time := current_period * ordi_size +
        (  past_history[i]  *  0.1  *  ordi_size  )
        else  due_time  :=  current_period  *  ordi_size  ;

        time_of_joining_the_queue  :=  0.0;
        operation_number  :=1;
```

```
        with  array_for_basic_data_about_asslys[i]  do
        begin
        with  array_of_processing_sequence[operation_number]  do
        begin
           current_mc_no := mc_no;
           pt:=setup_time  +  processing_time_per_unit  *  size;
        end;
        work_rem:=0.0;

        for  process_no  :=  1  to  no_of_processing_needed  do
        with  array_of_processing_sequence[process_no]  do
        work_rem := work_rem+ setup_time + processing_time_per_unit * ss ;
        (*     writeln(h,'work_remaining',work_rem:6:2);*)
        (*crr_value := 4000.00 /work_rem;
        writeln(h,'CRR   VALUE   ',crr_value:4:2);*)
        end;  (*  array  *)

        time_of_joining_the_queue  :=  time;
        next  :=  nil;
      end;(*  item  *)

        insert(item);

        if   array_for_process_mc[item^.current_mc_no].
        work_in_progress = false then
           load_the_mc(item^.current_mc_no,clock_time);

    end;
false   :   search_for_parent(i,ss,time);
end;  (*  case  for  needs  processing  end  *)

end;  (*  for  all  asslys  *)

procedure      update_the_item(item:ptr_to_lot;time:real);
var
pred_ptr,succ_ptr    :ptr_to_assly;
j,tt,ii,ss:integer;
mnj  :real;
```

```
begin
ii  :=2;
with item^ do
      begin
        work_rem := work_rem -pt;
        item^.operation_number := item^.operation_number +1;
        with array_for_basic_data_about_asslys [assly_no] do
        if item^.operation_number <= array_for_basic_data_about_asslys [assly_
        no_of_processing_needed then

        begin
          with array_of_processing_sequence[operation_number] do
          begin
            current_mc_no := mc_no;
            pt := setup_time + processing_time_per_unit * size;
          end;
        ii:=1;
        end (then )

      else

        begin
          inv_status[item^.assly_no]   :=
          inv_status[item^.assly_no]  -  item^.size  ;
          tt  :=  array_for_basic_data_about_asslys[assly_no].
          transformed_assly_no;

          ii:=  0;
         end; ( else )

       end ;( item)

     case ii of
     0 : begin

           ss  :=  item^.size;
           j := 1 ;
```

```
            while  inv_array[item^.assly_no].aa[j].planned_order  =  ss  do
            j := j +1 ;
            mnj := time - ( j + 1) * 4000.0;
            if  mnj  >  0.0  then  late_data[tt] := late_data[tt] +
            mnj  *  item^.size;
            waiting_time_during_processing  :=
                item^.total_wait_time;

            ss  :=  item^.size;
            dispose(item);
            initialise_the_new_item(tt,ss,time);
          end;
      1 :  begin
            item^.time_of_joining_the_queue  :=  time  ;
            insert(item);

            if  array_for_process_mc[item^.current_mc_no].work_in_progress  =  fal
            load_the_mc(item^.current_mc_no,clock_time);

          end;
      2  :;
      end;

end; { update the item }


procedure        sequencing_initialisations(init_time:real);

var
 j,aa,yy,process_no,ii,m      :integer;
 front:ptr_to_lot;
 item2,pred_ptr:ptr_to_assly;

begin   (*   sequencing_initialisations   *)

  (*  writeln(h,'SEQUENCING  INIT  AT    ',init_time:5:2);  *)

  for  i  :=  1  to  15  do
```

```
            while  inv_array[item^.assly_no].aa[j].planned_order  =  ss  do
            j := j +1 ;
            mnj := time - ( j + 1) * 4000.0;
            if  mnj > 0.0 then late_data[tt] := late_data[tt] +
            mnj  *  item^.size;
            waiting_time_during_processing  :=
                item^.total_wait_time;

            ss  :=  item^.size;
            dispose(item);
            initialise_the_new_item(tt,ss,time);
          end;
      1 :  begin
            item^.time_of_joining_the_queue  :=  time  ;
            insert(item);

            if  array_for_process_mc[item^.current_mc_no].work_in_progress  =  fal
            load_the_mc(item^.current_mc_no,clock_time);

          end;
      2 :;
      end;

end; { update the item }


procedure        sequencing_initialisations(init_time:real);

var
 j,aa,yy,process_no,ii,m      :integer;
 front:ptr_to_lot;
 item2,pred_ptr:ptr_to_assly;

begin   (*   sequencing_initialisations   *)

  (*  writeln(h,'SEQUENCING  INIT  AT    ',init_time:5:2);  *)

  for  i  :=  1  to  15  do
```

```
            while  inv_array[item^.assly_no].aa[j].planned_order  =  ss  do
            j := j +1 ;
            mnj := time - ( j + 1) * 4000.0;
            if  mnj > 0.0 then late_data[tt] := late_data[tt] +
            mnj  *  item^.size;
            waiting_time_during_processing  :=
                item^.total_wait_time;

            ss  :=  item^.size;
            dispose(item);
            initialise_the_new_item(tt,ss,time);
          end;
      1 :  begin
            item^.time_of_joining_the_queue  :=  time ;
            insert(item);

            if  array_for_process_mc[item^.current_mc_no].work_in_progress  =  fal
            load_the_mc(item^.current_mc_no,clock_time);

          end;
      2 :;
      end;

end; { update the item }


procedure        sequencing_initialisations(init_time:real);

var
 j,aa,yy,process_no,ii,m      :integer;
 front:ptr_to_lot;
 item2,pred_ptr:ptr_to_assly;

begin   (*   sequencing_initialisations   *)

  (*  writeln(h,'SEQUENCING  INIT  AT    ',init_time:5:2);  *)

  for  i  :=  1  to  15  do
```

```
  with array_for_process_mc[i] do
  if finish_time > init_time then
  begin
  if finish_time < simulation_time then
  utilisation_time := finish_time - init_time
                    else utilisation_time := simulation_time - init_time ;
  end
  else utilisation_time := 0.0;

  for i := 1 to 15 do
  with array_for_process_mc[i] do
  if ( finish_time < init_time ) and ( finish_time <> 0.0 ) then
       begin
         finish_time := 0.0;
         ptr_to_lot_being_processed := nil ;
         work_in_progress := false;
       end;

  yy := 0;
  for i := 1 to max_assly_no do
  begin
  case array_for_basic_data_about_asslys[i].no_of_child_asslys of
  0 :
      if inv_array[i].aa[current_period].planned_order > 0 then
      begin
  initialise_the_new_item(i,inv_array[i].aa[current_period].
  planned_order,init_time);
  yy := yy + 1;
  end;

  1:;   (* writeln(h,'it has only one child ');*)

2..max_child_assly  :;


end; (* case for no of child *)
end;
end;(* sequencing over *)
```

```
procedure  start_the_process;
begin
for i := 1 to max_mc_no do begin
                              load_the_mc(i,0.0);
                              end;
end;{ start_the process }


procedure   event(time:real);
var
order,n,m   :integer;
init_over   :boolean;


begin
  m:=1;
  for i:= 1 to max_mc_no do
  with array_for_process_mc[i] do
  if ptr_to_lot_being_processed <> nil then
  begin
    if time = finish_time then
    begin
      m := 0 ;
      item := ptr_to_lot_being_processed;
      ptr_to_lot_being_processed := nil;
      work_in_progress := false ;
      finish_time := 0.0;
      load_the_mc(i,time);
      update_the_item(item,time);
    end;
  end;

  if m = 1 then
  for i := 16 to 20 do
  with array_for_assly_mc[i] do
  if finish_time = time then
                          begin
```

```
                    (*    writeln(oo,assly_no_being_assembled,
                    '    assly  oprn  over');*)
                    finish_time  :=  0.0;
                    work_in_progress  :=  false;

                    init_over  :=  false;
                    order  :=  0;
                    n  :=  1;
                    repeat

                    if  demand[assly_no_being_assembled,n] > 0 then

                    begin
                      order  :=  n;
                      init_over  :=  true  ;
                    end;

                    n:=  n+1;
                    until ( init_over = true ) or ( n = current_period +

                    if ( ( order > 0 ) and ( init_over = true )) then

                    begin
                      current_lot_size  :=  current_lot_size  -
                              demand[assly_no_being_assembled,order];

                      demand[assly_no_being_assembled,order]  :=  0;
                    end;

                    initialise_the_new_item(assly_no_being_assembled,
                          current_lot_size,time);
                  end;
end;

procedure    update_clock_time(time:real);
var
init_over    :boolean;
nn:real;
```

```
begin (* update_clock_time *)
      i:=1;
      nn:= 0;
      next_event_at := 0.0 ;
      init_over:=false;
      repeat
      if  array_for_process_mc[i].finish_time  >  0.0  then
        begin
       nn   :=   array_for_process_mc[i].finish_time;
       init_over  :=  true;
       end;
       i:=   i+1;
       until  (init_over)  or  (i  >  max_mc_no);

      if  init_over  =  false  then
      repeat
      if  array_for_assly_mc[i].finish_time  >  0.0  then
      begin
      nn   :=   array_for_assly_mc[i].finish_time;
      init_over  :=  true;
      end;
      i:=   i+1;
      until  (init_over)  or  (i  >  20);

      if  (  nn  =  0.0  )  then  clock_time  :=  simulation_time+100
                        else
                        begin
      (*  writeln('init  event  at       :',nn:5:2);  *)
      next_event_at  :=  nn;

      for i := 1 to max_mc_no do
      begin
      if  (  array_for_process_mc[i].finish_time  >  time  )  and
      (array_for_process_mc[i].finish_time  <  next_event_at)  then
      next_event_at   :=   array_for_process_mc[i].finish_time;
      end;
```

```
      for i := 16 to 20 do
      begin
      if (array_for_assly_mc[i].finish_time > time) and
      (array_for_assly_mc[i].finish_time < next_event_at) then
      next_event_at := array_for_assly_mc[i].finish_time;
      end;

      clock_time := next_event_at;

      end;

end; (* update_clock_time *)


procedure do_sequencing;
var
check_time :real;
item2 :ptr_to_assly;
j,m :integer;
ini_found :boolean;

begin (* do_sequencing *)

      (* initialisation of array_for_process_mc *)

  for i := 1 to max_mc_no do
  with array_for_process_mc[i] do
  begin
    work_in_progress := false;
    utilisation_time := 0.0;
    finish_time := 0.0;
    no_of_jobs_waiting := 0;
    lot_ptr := nil;
    ptr_to_lot_being_processed := nil;
  end;

  for i := 16 to 20 do
  with array_for_assly_mc[i] do
```

```
begin
  current_lot_size  :=  0;
  assly_no_being_assembled := 0;
  assly_ptr  :=  nil;
  finish_time  :=  0.0;
  work_in_progress  :=false;
end;

  lot_no  :=  0;

  current_period  :=  1;
  for i := 1 to max_assly_no do
  if  array_for_basic_data_about_asslys[i].no_of_child_asslys  >  1  then
  begin
          new(item2);
          with  array_for_basic_data_about_asslys[i]  do
          with item2^ do
          begin
            late_ness        := 0.0;
            order_completed := 0;
            current_mc_no := mc_no_for_assly;
            assly_no   :=   array_for_basic_data_about_asslys[i].assly_no;
            no_of_childs   :=   no_of_child_asslys;
            j:=1;

            ini_found  :=  false  ;
            repeat
              if  (  inv_array[assly_no].aa[j].planned_order  >  0  )  then

                  begin
                    early_assly_order  :=  j  ;
                    ini_found  :=  true  ;
                  end;

                  j  :=  j  +1;

                until ((  ini_found  =  true  )  or  (  j  =  max_period_no  +  1  )  );
```

```
for j := 1 to no_of_childs do
begin
    child_infor[j].no    :=    array_of_child_asslys[j].
    child_assly_no;
    child_infor[j].waiting_time   :=   0.0;

    for m := 1 to max_period_no do
    begin
      child_infor[j].arrival_time[m]   :=   0.0;
      child_infor[j].qty_arrived[m]   :=   0;
      child_infor[j].qty_needed[m]   :=   0;
      child_infor[j].lot_number[m]   :=   0;
      if  inv_array[assly_no].aa[m].planned_order  >  0  then
        child_infor[j].qty_needed[m]    :=
        array_of_child_asslys[j].qty_to_be_contributed
        *    inv_array[assly_no].aa[m].planned_order
        else  child_infor[j].qty_needed[m]  :=  -1  ;

    end;


    m := 1;
    ini_found  :=  false  ;
    repeat
      if  (  child_infor[j].qty_needed[m]  >  0  )  then

      begin
        child_infor[j].early_demand  :=  m  ;
        ini_found  :=  true  ;
      end;

      m := m +1;

    until  (  ini_found  =  true  )  or  (  m  >  max_period_no  );

end;

next  :=  nil;
```

```
end; { item }

insert_in_assly_mc(item2);

end; { if it is a assembly  }

clock_time := 0.0;
continue := false;



while ( current_period < 41 ) do

begin

  (* writeln('CURRENT PERIOD    :',current_period); *)
  initial_time := (current_period - 1 ) * 4000.00;
  simulation_time := current_period * 4000.00;

  sequencing_initialisations(initial_time);

  for i := 1 to max_mc_no do load_the_mc(i,initial_time);

  update_clock_time(initial_time);

  while(clock_time <= simulation_time) do

  begin
    event(clock_time);
    update_clock_time(clock_time);
  end;

  (*   writeln(mn,'current   period',current_period:5);
  writeln(mn,'      mc no          ','    utilisation_rate      ','jobs w
  for i := 1 to max_mc_no do
  with array_for_process_mc[i] do
  writeln(mn,'    ',i:4,'                          ',utilisation_time/4000.
  '                          ',finish_time:8:2,'   ',no_of_jobs_waiting:5
```

```
                for i := 1 to max_assly_no do
                  begin
                    (*   writeln(inv_status[i]);   *)
                    cost := cost + ( inv_status[i] *
                            inv_array[i].aa[current_period].inv_carry_cost      );
                    actual_cost[i] := actual_cost[i] + ( inv_status[i] *
                            inv_array[i].aa[current_period].inv_carry_cost      );
                  end;

                (*   writeln('cost',cost:10:2);   *)

                current_period := current_period + 1;
                continue := false ;

                for i := 1 to max_mc_no do
                if array_for_process_mc[i].utilisation_time < 4000.00 then
                umi := umi + array_for_process_mc[i].utilisation_time/4000.00 ;
              end; { while }

    end;   (* do_sequencing ends *)


procedure          collect_and_print_statistics;
var
pred_ptr   :ptr_to_assly;
j,yy,yyy,cc    :integer;
min,max  :real;

begin   (*  collect_and_print_statistics  *)

writeln(oo,'statistics');

for i := 1 to max_assly_no do priority[i] := 0;

for i := 16 to 20 do
with array_for_assly_mc[i] do
```

```
begin
  (* writeln('ass mc no ',ii); *)
  pred_ptr := assly_ptr;
  j:=0;
  while pred_ptr <> nil do

  begin
    with pred_ptr^ do

    begin
       (* writeln(ooo,'ass no ',pred_ptr^.assly_no); *)
       min := 1000000000.00;
       max := 0.0 ;
       yy := 0 ;
       yyy := 0;
       for j := 1 to no_of_childs do

       begin
         if child_infor[j].waiting_time > max then
            begin
              yy := child_infor[j].no ;
              max := child_infor[j].waiting_time ;
            end;

         if child_infor[j].waiting_time < min then
            begin
              yyy := child_infor[j].no ;
              min := child_infor[j].waiting_time ;
            end;

       end;  { all child over }

       stock_out := stock_out + order_completed ;

       case pred_ptr^.assly_no of
       1,5,19,27 : begin
                     if order_completed <> 0 then
```

```
                    begin
                    late_ness   /order_completed/100000:8:2);
                    end_late := end_late + ( late_ness /order_completed);
                    end else end_late := end_late + late_ness ;
                  end;
2..4,6..18,20..26,28..max_assly_no      :;
end;
late_data[pred_ptr^.assly_no]  :=  late_ness  ;

if yy <> 0 then
begin
priority[yy]  :=  priority[yy]  +  1  ;
with  array_for_basic_data_about_asslys[yy]  do
if  no_of_child_asslys  >  0  then
begin
  for  j  :=  1  to  no_of_child_asslys  do
  begin
    with  array_of_child_asslys[j]  do
    begin
      cc:=array_for_basic_data_about_asslys[assly_no].
      array_of_child_asslys[j].child_assly_no;
      priority[cc]  :=  priority[cc]  +  1  ;
    end;
  end;
end;
end;

if yyy <> 0 then
begin
priority[yyy]  :=  priority[yyy]  -  1;

with  array_for_basic_data_about_asslys[yyy]  do
if  no_of_child_asslys  >  0  then
begin
  for  j  :=  1  to  no_of_child_asslys  do
  begin
    with  array_of_child_asslys[j]  do
    begin
```

```
                cc:=array_for_basic_data_about_asslys[assly_no].
                array_of_child_asslys[j].child_assly_no;
                priority[cc] := priority[cc] - 1 ;
              end;
            end;
          end;
          end;

      end;

      pred_ptr := pred_ptr^.next;
    end;
  end;

  for i := 1 to max_assly_no do
  with array_for_basic_data_about_asslys[i] do
  if no_of_child_asslys > 1 then
  aggre_late[iteration_no] := aggre_late[iteration_no] + late_data[i] ;

  umi := umi / ( max_period_no * max_mc_no );
  for i := 1 to max_assly_no do
  case priority[i] of
  0 :;
  1..50:past_history[i] := past_history[i] + 1 ;
  -50..-1 : past_history[i] := past_history[i] -1;
  end; { case }

  if ( heuristic = 2 ) and ( iteration_no > 1 ) then
  if best_late > aggre_late[iteration_no] then
  begin
  best_late := aggre_late[iteration_no] ;
  best_iteration := iteration_no;
  end;

end; (* collect_and_print_statistics *)

procedure do_heuristic_two;
var
```

```
i,iii,old_lots,no,new_lots       :integer;
smt,ratio,max   :real;
stop    :boolean;
old     :array[1..max_period_no] of integer;

begin
  r := [ ];
  for i := 1 to max_assly_no do
  begin
    diff[i] := ( actual_cost[i] ) - there_cost[i] ;
    if (not(i in s ) ) then
                                  begin
                                    ratio := diff[i] / there_cost[i] ;
                                    if ( ratio > 0.1 ) then r := r + [i] ;
                                  end;
  end;

  max := 0.00;
  no := 0;

  for i := 1 to max_assly_no do
  if ( i in r ) and (not(i in s )) then
  if diff[i] > max then begin
                              max := diff[i];
                              no  := i ;
                        end;
  stop := false ;
  iii := 0;
  repeat
    iii := iii + 1;
    for i := 1 to max_period_no do inv_array[no].aa[i].inv_carry_cost :=
                      inv_array[no].aa[i].inv_carry_cost * 1.2 ;
    old_lots := 0 ;
    new_lots:=0;
    smt := 0;
    for i := 1 to max_period_no do
    begin
    if  inv_array[no].aa[i].lot_quantities > 0 then
```

```
        begin
          old_lots := old_lots + 1;
          smt := smt + inv_array[no].aa[i].ordering_cost;
        end;
    if inv_array[no].aa[i].net_req > 0 then new_lots := new_lots + 1;
    old[i] := inv_array[no].aa[i].lot_quantities ;
    end;
    no_of_setup := no_of_setup - old_lots ;
    inven_cost := inven_cost - ( there_cost[no] - smt );
    set_up_cost := set_up_cost - smt ;
    total_cost := total_cost - there_cost[no];
     writeln('old',old_lots,'total        net',new_lots);
     old_lots := new_lots;
     do_lot_sizing(no);
     new_lots := 0;
     for i := 1 to max_period_no do
     if inv_array[no].aa[i].lot_quantities > 0 then new_lots := new_lots + 1;
     for i := 1 to max_period_no do
     if old[i] <> inv_array[no].aa[i].lot_quantities then stop := true ;
     writeln('new',new_lots);
     no_of_setup := no_of_setup + new_lots;
  until ( stop = true ) or ( new_lots >= old_lots );
  k[no] := k[no] + 1 ;
  if k[no] >= 2 then s := s + [no];

end;

procedure mrp_system;
begin
    ordi_size := 4000.00;
    iteration_no := 0;

    for i := 1 to max_assly_no do
      begin
        priority[i] := 0;
        past_history[i] := 0;
        late_data[i] := 0.0;
        actual_cost[i] := 0.0;
```

```
        k[i] := 0;
      end;

    for i := 1 to 20 do aggre_late[i] := 0.0 ;

    r := [ ];
    s := [ ];

repeat

    umi := 0.0 ;
    cost :=0.0;
    stock_out := 0;
    end_late := 0.0 ;
    no_of_stockout:=0;
    for i := 1 to max_assly_no do
    begin
      late_data[i] := 0.0;
      actual_cost[i] := 0.0;
      diff[i] := 0.0;
    end;
    iteration_no := iteration_no + 1;
    writeln('ITERATION NO :',iteration_no);
    writeln(ooo,'ITERATION NO :',iteration_no);

    iterations_can_continue :=false;
    while not(iterations_can_continue) do
    begin
      do_sequencing;
      iterations_can_continue := true;
    end;
    collect_and_print_statistics;
    if heuristic = 2 then do_heuristic_two;

  {  disposing all asslys }

    for i := 16 to 20 do
    with array_for_assly_mc[i] do
```

```
    if assly_ptr <> nil then
    begin
      pred := assly_ptr;
            repeat
      succ := pred^.next;
      if pred <> nil then dispose(pred);
      pred := succ;
      until ( succ = nil );
    end;

    for i := 1 to max_mc_no do
    with array_for_process_mc[i] do
    if lot_ptr <> nil then
    begin
    item := lot_ptr;
      repeat
      base := item^.next;
      dispose(item);
      item := base;
      until ( base = nil );
    end;

    until ( iteration_no = iteration_required );
    if heuristic = 2 then writeln(ooo,'best sol late :',best_late:10:2,
                                        'iteration no ',best_iteration:4);

end;

begin (* main *)

    assign(f,'o16');
    rewrite(f);
    assign(h,'ss');
    rewrite(h);
    assign(oo,'result.dat');
    rewrite(oo);
    assign(mn,'result2.dat');
    rewrite(mn);
```

```
assign(kk,'result3.dat');
rewrite(kk);
assign(ooo,'result4.dat');
rewrite(ooo);
read_input_data;
initialisation;
write('ENTER CHIOCE OF HEURISTIC NO (1/2/3) :');
read(heuristic);

mrp_initialise;
read_assly_data;

best_late := 100000000000.00;
best_iteration := 0;
case heuristic of
1,2 : begin
        iteration_required := 20;
        mrp_system;
      end;
 3  :begin
       heuristic := 2 ;
       iteration_required := 20;
       mrp_system;
       initialisation;
       read_input_data;
       mrp_initialise;
       read_assly_data;
       iteration_required := best_iteration;
       heuristic := 2;
       mrp_system;
       heuristic := 1;
       iteration_required := 20;
       mrp_system;
     end;
 end;
close(f);
close(h);
close(oo);
```

```
        close(mn);
        close(kk);
        close(ooo);
end.
```